TRABAJO FIN DE GRADO:

DESARROLLO DE LIBRERÍA OPEN-SOURCE PARA EL PROYECTO SGPS

Autor: Isaac Rivero Guisado
Tutor: Luis Enrique Moreno Lorente
Director del proyecto: Javier Victorio Gómez González

Índice

Índice de figuras

Resumen

Conocer la ubicación exacta de donde se encuentra una persona o algún objeto es muy importante, ya que esto puede ayudar en múltiples tareas cotidianas, como por ejemplo saber llegar a algún lugar del que solo se conoce la calle. También puede colaborar en tareas mucho más importantes, como la prevención de accidentes en aviación o incluso el rescate de personas perdidas o incapacitadas temporalmente debido a un accidente.

El sistema de posicionamiento global o GPS es un sistema de navegación por satélite (GNSS) que permite ubicar en todo momento donde se encuentra un objeto o una persona. El funcionamiento básico de este tipo de sistema consiste en la trilateración de la posición mediante satélites colocados en la órbita de la Tierra, lo cual hace que el dispositivo utilizado sea dependiente de muchos factores.

En este trabajo se presenta un conjunto de librerías implementadas en C++ que servirán de apoyo para una alternativa a este sistema de posicionamiento global. En lugar de utilizar satélites para trilaterar una posición, se valdrá de la intensidad de luz que emite el sol en un determinado lugar.

Abstract

To know the exact location where someone or something is placed is really important because it is useful for a lot of quotidian tasks as, for example, to reach a stree from the current position, to prevent airplane crashes or search and rescue operations.

The Global Positioning System (GPS) is global navigation satellite system (GNSS) that allows to locate an object or person at any time. The working principle of these systems consists of the trilateration of location by satellites situated on the Earth's orbit. Therefore a GPS devide depends on many factors.

This work presents a library implemented in C++ that will support an alternative for this Global Positioning System. It will use the sunlight intensity at a certain place instead of using satellites.

1. Introducción

El Proyecto SGPS (Sunlight intensity-based Global Positioning System) encuentra un sistema de posicionamiento global basado en las características que ofrecen los rayos del sol al llegar a la Tierra. Este documento contiene el diseño de una librería para la utilización de este sistema junto con las bases necesarias para comprender su funcionamiento.

1.1. Motivación

Los sistemas de geolocalización actuales permiten conocer una ubicación con escasos metros de error, pero tienen algunas desventajas importantes: son complejos, caros y necesitan la autorización del fabricante para su uso, lo que los hace altamente dependientes. Además, se necesita en todo momento que el GPS esté conectado con algunos satélites cercanos, lo cual hace que estén sujetos a factores externos como las tormentas solares. La emanación de partículas energéticas procedentes de las tormentas pueden inutilizar los satélites de los que se vale el GPS; si la tormenta es fuerte, puede incluso ocasionar la pérdida de control de los mismos. Otro problema existente es la acumulación de "basura espacial", que ocupa gran parte del espacio orbital de los satélites.

En Mayo de 2010 Frode Eika Sandnes ideó un algoritmo para localizar la ubicación de un objeto mediante las condiciones de luz de una foto, la fecha y hora de la realización de la misma[1]. Tras esto, Javier V. Gómez comenzó a trabajar en la idea de Frode Eika acerca de un sistema capaz de localizar la posición mediante intensidad de luz[2].

El diseño de esta librería servirá como base para futuras ideas y ayudará a su desarrollo: por ejemplo, incluir en el algoritmo otras magnitudes como la presión atmosférica o la temperatura, intentar prever el error de latitud que se produce en algunos casos o diseñar una forma de aprendizaje que mejore la precisión para encontrar la salida y la puesta de sol.

1.2. Objetivos

Los objetivos principales de este proyecto son los siguientes:

- Crear las librerías necesarias en C++ para la visualización y manejo del algoritmo detallado en [2].
- Buen diseño de las librerías para facilitar la extensión.
- Creación de herramientas adicionales para facilitar la compresión de las librerías.
- Adaptar las librerías para que sean open source, es decir, que sea un código abierto, en el que cualquier persona pueda trabajar y mejorar.
- La librería diseñada debe servir de base para futuros trabajos con SGPS.

1.3. Estructura

El presente documento está dividido en siete secciones. Tras la introducción, se explican brevemente los fundamentos necesarios para conocer el funcionamiento del sistema SGPS. Más adelante, en el tercer capítulo, se detallan las tecnologías que sirven de ayuda al diseño del proyecto.

Los capítulos cuarto y quinto están dedicados al desarrollo y las pruebas del proyecto: se explican la librería propuesta, los problemas que ha habido para su diseño y algunos ejemplos para comprobar su funcionamiento.

El sexto capítulo sirve de información adicional para que cualquier persona interesada pueda desarrollar la librería. Y para concluir el proyecto, el séptimo y último capítulo indica los futuros trabajos que puedan surgir a partir de este.

2. Fundamentos teóricos

En esta sección se explicarán varios conceptos importantes que ayudarán a la compresión del resto del documento: desde los parámetros físicos en los que se basa el proyecto hasta una exposición del funcionamiento del SGPS.

2.1. Conceptos básicos

El SGPS se basa principalmente en la medida de intensidad de luz de un lugar para conocer su ubicación, aprovechando que los rayos del sol llegan de una forma distinta en cada momento y en cada punto del planeta. Como se puede ver en la figura 1, la luz llega de frente a la altura del sur de África y con mucha inclinación en Europa con lo que la intensidad de luz en África es completamente distinta a la de Europa.

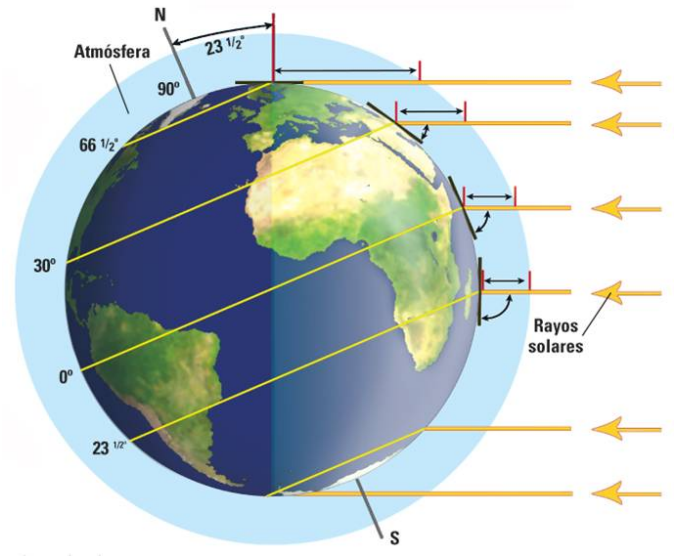

Figura 1: Como llegan los rayos del sol a la Tierra. Imagen cortesía de Julia Gracia

Como se ve en la figura 2 la Tierra está divida en paralelos y meridianos. Un paralelo es un círculo imaginario perpendicular al eje de la Tierra. Un meridiano es el semicírculo imaginario sobre la superficie terrestre, que pasa a través del polo norte y del polo sur, cortando los paralelos con un ángulo recto.[3].

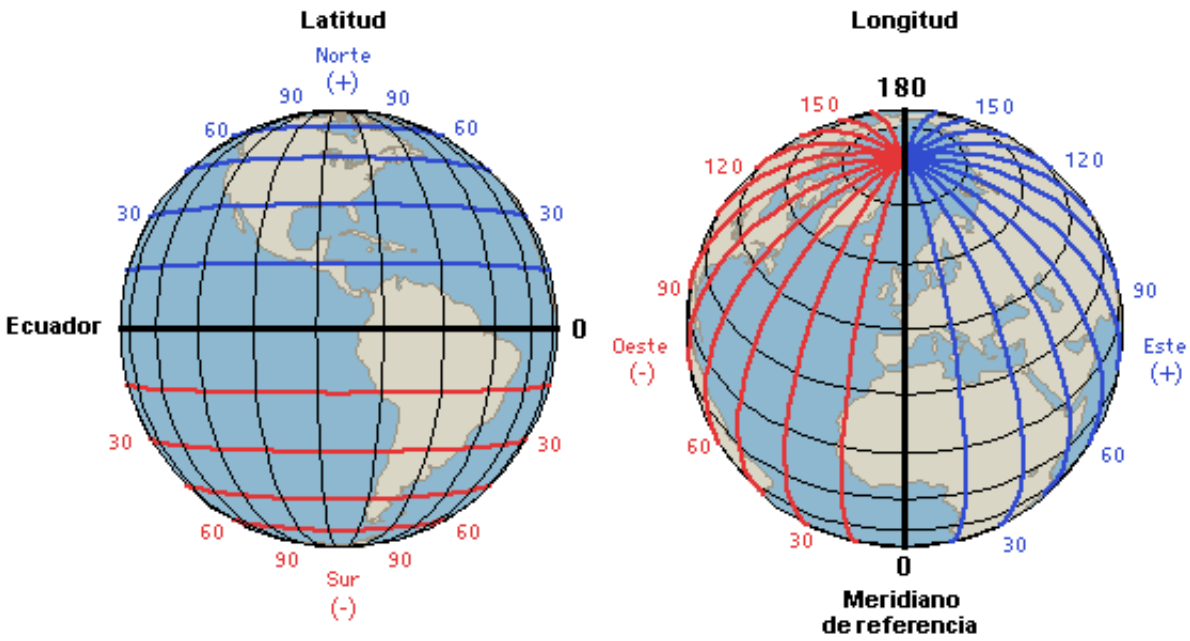

Figura 2: Meridianos y paralelos. Imagen cortesía de *Picasa Web*

Los paralelos y meridianos conforman la base de un sistema de referencia conocido como coordenadas geográficas, que determina los ángulos laterales de la superficie terrestre[4], mediante dos coordenadas angulares: latitud y longitud. Estos parámetros se basan en el sistema de coordenadas esféricas, que determina la posición espacial de un punto mediante una distancia y dos ángulos[5]. Como se puede ver en la figura 3, 'A' queda totalmente representado por el radio r, que en caso de ser el planeta Tierra sería su radio, y los ángulos ψ y φ.

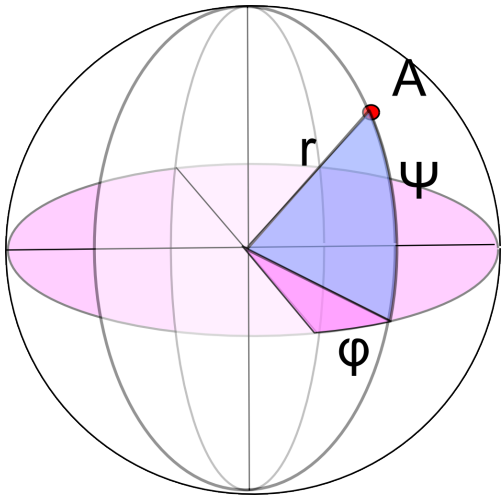

Figura 3: Ejemplo de coordenadas esféricas. Imagen cortesía de *Pawean*

La latitud de un punto de la superficie terrestre es la medida (en grados) del ángulo que hay entre la recta que une ese punto con el centro de la Tierra y el plano del ecuador[6], por lo que en la figura 3 la latitud equivaldría al ángulo ψ.

La longitud es el ángulo (en grados) que indica la posición este-oeste de un punto sobre la superficie terrestre a partir del meridiano de Greenwich[7]. En la figura la longitud sería el ángulo φ.

2.2. Funcionamiento del SGPS

Como ya se ha comentado, el SGPS (Sunlight intensity-based Global Positioning System) localiza objetos exteriores sólo por la intensidad de luz. El sistema descrito depende de la estacionaridad, lo que significa que el objeto tiene que estar inmóvil durante todo el día, o al menos durante el tiempo de luz diurna. Sin embargo, debido a la precisión del sistema y a que la intensidad de la luz solar puede considerarse constante en áreas no muy grandes, el sistema puede funcionar correctamente aunque el objeto realice pequeños movimientos.

2.2.1. Requisitos del sistema

El método de geolocalización se ha diseñado para hacerlo lo más simple posible. Sus componentes son un sensor de iluminación, un microprocesador y un reloj incorporado que mantenga una cuenta relativamente precisa de la hora y la fecha.

Para el reloj, hay que tener en cuenta que el objeto puede preguntar en cualquier momento la hora actual t y la fecha d, lo que hace necesario el uso de un dispositivo que proporcione ambos parámetros. Bastaría con un reloj digital, que además tiene la ventaja de tener una autonomía de varios años. El tiempo se ajusta a la hora universal (UTC) para que los cálculos sean más simples, y está representado en forma decimal en el intervalo de 0 a 24 evitando problemas por el horario de verano o la diferencia horaria entre países. La fecha d se representa mediante el día del año, donde 01 de enero es el día 1 y así sucesivamente. Respecto al sensor de iluminación, valdría con un sensor de intensidad de luz, que da un voltaje directamente proporcional a la intensidad de la luz incidente. Este tipo de sensores no consumen energía eléctrica, ya que transforma la energía luminosa que recibe en energía eléctrica sin ningún tipo de fuente. También puede emplearse un medidor del valor de exposición, dispositivo que emplean las cámaras para variar la ganancia de la imagen en función de la luz incidente. El microprocesador debe analizar los datos suministrados por el sensor, y consumirá la mayor parte de la energía del sistema.

La operación SGPS se resume en el diagrama de flujo presentado en la siguiente figura.

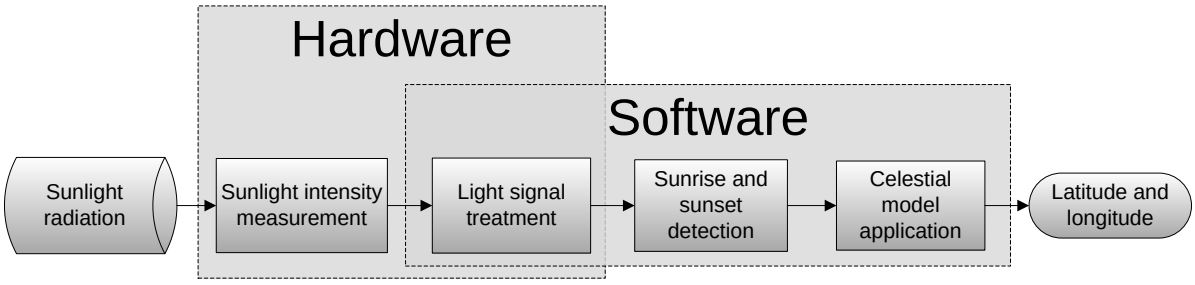

Figura 4: Diagrama de flujo del SGPS

El sistema puede ser diseñado de una manera más compleja, incluyendo otros

dispositivos de medición o el uso de un microprocesador más potente. En la siguiente figura se sugiere un implementación en Arduino.

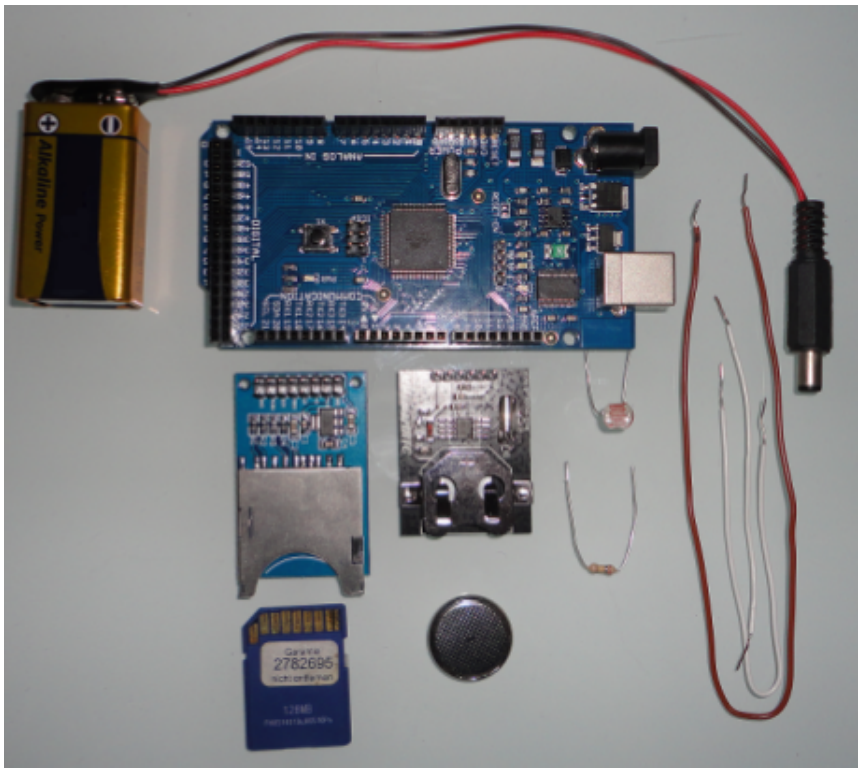

Figura 5: Implementación en Arduino

2.2.2. Celestial Model

El SGPS usa un modelo llamado *Celestial Model* que localiza objetos al aire libre usando sólo datos de intensidad de la luz solar para un lugar determinado. Con estos datos es posible obtener la salida y puesta del sol en ese lugar; sólo con estos dos valores, *Celestial Model* determina las coordenadas de longitud y latitud. Por lo tanto, las coordenadas se puede expresar como una función de la salida y la puesta de sol, como se ve en la siguiente figura.

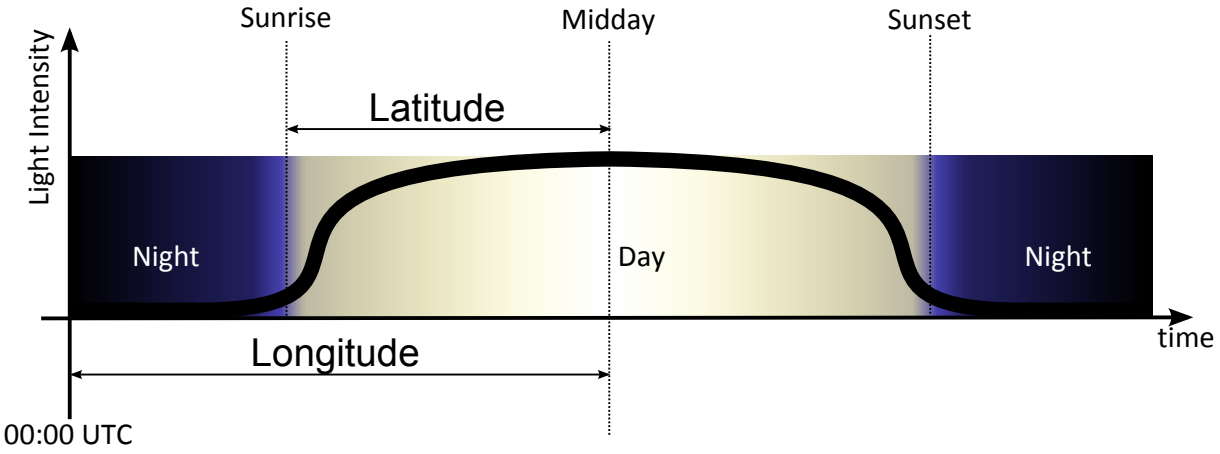

Figura 6: Celestial Model

Además de este modelo, existe un algoritmo exacto que es capaz de conocer la salida y puesta del sol en un lugar específico con sólo conocer sus coordenadas y la fecha. Sin embargo, no es posible realizar el procedimiento inverso y obtener las coordenadas a través de la salida y la puesta de sol, ya que algunas fórmulas dependen de la latitud y longitud, y al tratar de obtener el resultado se observa que la latitud depende de la longitud y viceversa.

El problema planteado por SGPS se puede reducir a la identificación de los tiempos de la puesta y la salida del sol para un día concreto. Dada una medición exacta de la hora de la salida del sol $t_{sunrise}$ y de la puesta de sol t_{sunset}, el mediodía solar t_{midday} es simplemente:

$$t_{midday} = \frac{t_{sunrise} + t_{sunset}}{2}$$

Al aplicar esta ecuación hay que tener en cuenta que la representación UTC de tiempo asume que la salida del sol se produce antes de la puesta de sol. Si se produce la puesta del sol antes de la salida del sol dentro de la ventana de tiempo 24 horas UTC, se trata de un día fraccionado. En este caso, la salida del sol ocurre el día anterior a la puesta del sol. Hay que tener esto en cuenta, ya que si la ecuación se aplica directamente con un tiempo de puesta de sol anterior a la salida, la hora del mediodía no será correcta; el resultado será la medianoche. Para subsanar este error, se suman 12 horas al resultado de la ecuación.

A continuación se explicará el algoritmo realizado para obtener la longitud y latitud. De aquí en adelante se tomará el tiempo en formato decimal de 0 a 24 horas y los ángulos en radianes.

La puesta de sol angular se puede calcular de la siguiente manera:

$$\alpha_{midday} = \frac{\pi}{12} * |t_{sunrise} + t_{sunset}|$$

y la declinación del sol δ puede ser aproximada por la siguiente serie de Fourier:

$$\delta = 0{,}006918 - 0{,}399912 * cos(\beta) + 0{,}070257 * sin(\beta)$$
$$-0{,}006758 * cos(2\beta) + 0{,}000907 * sin(2\beta)$$
$$-0{,}002697 * cos(3\beta) + 0{,}00148 * sin(3\beta)$$

donde β es la fracción de año expresado en radianes dado por:

$$\beta = \frac{2\pi}{365} * d$$

y d es el día de 1 a 365. En el caso de un año bisiesto, d será 366.

Por último, se puede obtener las coordenadas. Para la longitud λ, donde los valores positivos representan el este y los valores negativos representan al oeste:

$$\lambda = 2\pi * \frac{12 - t_{midday}}{24}$$

y la latitud φ de la ubicación de los objetos se puede encontrar resolviendo numéricamente la siguiente ecuación, donde los valores positivos representan el norte y los valores negativos representan el sur:

$$cos(\alpha_{sunset}) = \frac{sin(-0{,}0145) - sin(\delta) * sin(\varphi)}{cos(\delta) * cos(\varphi)}$$

Las ecuaciones proporcionadas en esta sección permiten averiguar las coordenadas de objetos exteriores a través de los tiempos de la salida y la puesta del sol de un día dado.

2.2.3. Mejoras

Como futuras vías de trabajo, es posible realizar una serie de mejoras al proyecto. Los amaneceres se pueden predecir si se toman las primeras medidas, ya que la intensidad de la luz aumenta gradualmente durante un intervalo de tiempo antes de que comience la salida del sol. Si se toma una medida de la intensidad de luz que está por encima del valor de referencia de la noche, pero todavía por debajo de la salida del sol, es un signo de que esta se aproxima pronto y la frecuencia de muestreo se puede aumentar dinámicamente. Esto se ilustra en la figura 7, que muestra un gráfico de la intensidad auténtica obtenida usando una webcam. Es evidente que la intensidad se eleva durante aproximadamente 20 minutos antes de la salida del sol.

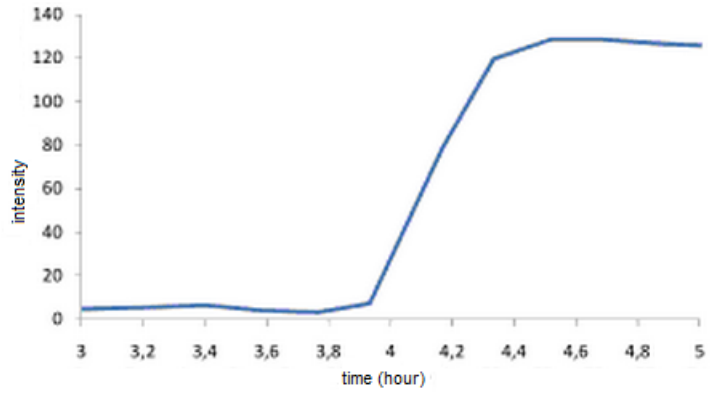

Figura 7: Incremento gradual de la intensidad cerca de la salida del sol

Sin embargo, puede ser más difícil obtener una previsión de una puesta de sol de esta manera, ya que la intensidad de luz puede caer de repente por la acción de alguna nube. La mejor manera de solucionar esto sería capturando información sobre el espectro de color y utilizándola para predecir mejor los amaneceres y puestas de sol. No todos los sensores pueden hacer esto, pero algunos, como los CCD de las cámaras digitales son capaces de detectar color, así como la intensidad. Otro método de predicción de atardeceres y amaneceres está basado en los grandes cambios en la tonalidad que se producen en los mismos. Esto se ilustra en la imagen 8 que presenta un tramo de 24 h obtenidas usando una cámara web.

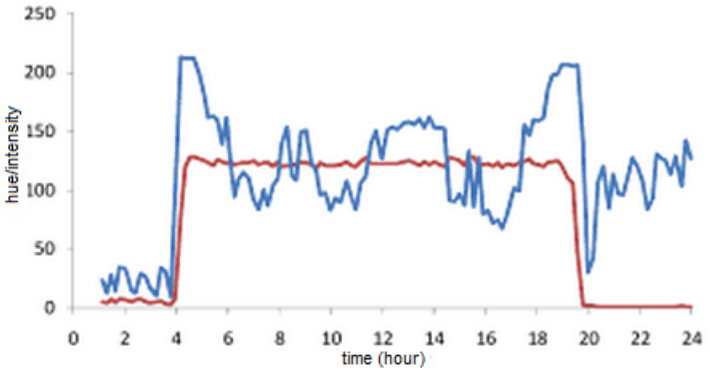

Figura 8: Cambios bruscos de tono predicen amaneceres y puestas de sol. Línea roja: intensidad de la imagen. Línea azul: tono de la imagen

La tonalidad h (r, g, b) de cada píxel se calcula a partir de los componentes R, G y B (rojo, verde y azul) de la siguiente manera:

$$h(r, g, b) = atan2(2r - g - b, \sqrt{3}(g - b))$$

La línea roja muestra la intensidad general de la imagen y la línea azul ilustra la tonalidad. Se pueden observar los dos picos en la curva de tonalidad, que señalan el cambio drástico de tono justo antes del amanecer y el atardecer.

3. Tecnologías asociadas

En esta sección se realiza una introducción a todo el área tecnológica sobre el que se basa el trabajo desarrollado. Se comenzará con el lenguaje C++, siguiendo por la herramienta para hacer gráficos llamada gnuplot, y terminando con las herramientas de desarrollo utilizadas, tales como Cmake o la colección de compiladores GNU.

3.1. C++

C++ se trata de un lenguaje de programación muy extendido y de gran potencia y flexibilidad. Un lenguaje de programación no es más que un idioma artificial diseñado para expresar acciones que tras un proceso de compilación, en el caso de que sea un lenguaje compilado, puedan ser entendidas por una computadora.

C++ fue creado en los años ochenta por Bjarne Stroustrup[8], extendiendo el antiguo pero potente lenguaje C a la programación orientada a objetos. Actualmente existe un estándar de C++, llamado ISO C++, al que se han ido añadiendo creadores de compiladores.

3.2. Gnuplot

Gnuplot es un programa de línea de comandos que puede generar gráficas de dos y tres dimensionales de funciones, datos y ajuste de datos. Es compatible con los sistemas operativos más populares (Linux, UNIX, Windows, Mac OS X).

El origen de gnuplot data de 1986. Puede producir sus resultados directamente en pantalla, así como en multitud de formatos de imagen, como PNG, EPS, SVG, JPEG, etc. Se puede usar interactivamente o en modo por lotes (batch), usando scripts[9]. A pesar de su nombre, este software no se distribuye bajo la licencia GPL de GNU; tiene su propia licencia de código abierto más restrictivo.

En la imagen 9 se puede ver un ejemplo de este programa en el que se pide que se dibuje la gráfica de un seno. Como se puede observar primero se hace una llamada al programa desde el terminal. Una vez iniciado, tan sólo con poner el comando *plot* y el nombre de la función el programa la dibuja.

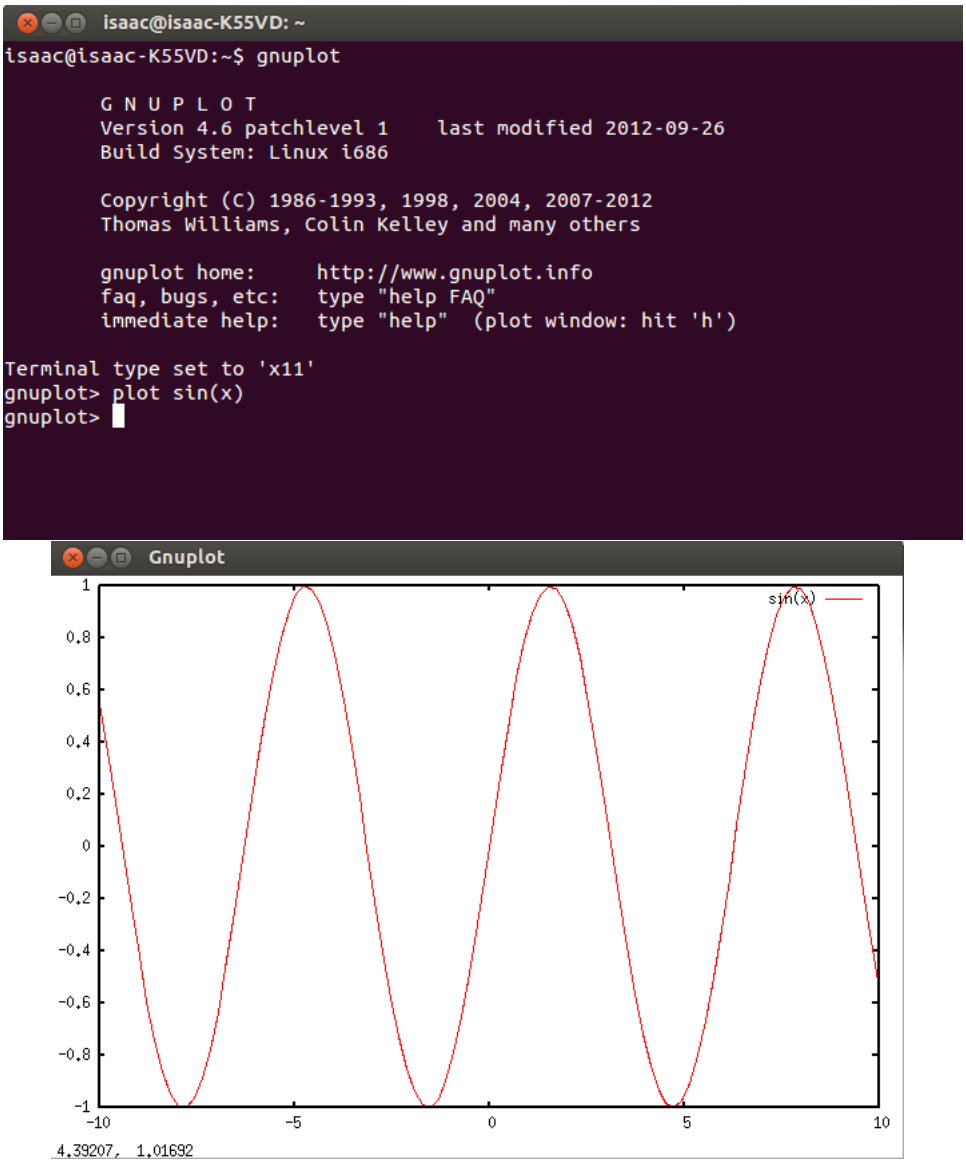

Figura 9: Ejemplo de gnuplot

3.3. Herramientas de desarrollo

En esta sección se hará un repaso de las herramientas utilizadas para poder programar en el lenguaje C++, como la colección de compiladores GCC o la herramienta de gestión de proyectos de software multiplataforma Cmake.

3.3.1. Colección de compiladores GNU

Figura 10: Logo de GCC

La colección de compiladores GNU (GNU Compiler Collection), GCC, son unas herramientas libres creadas por el proyecto GNU de forma libre y liberadas bajo la licencia GPL. Estos compiladores son básicos en los sistemas basados en UNIX, tanto libres como propietarios (como es el caso del Mac OS X de Apple).

GCC nació como parte del proyecto GNU, creando un compilador exclusivamente de C. Pese a ello, con el tiempo se fue extendiendo a más lenguajes, entre los que se encuentra C++ o Python. El proyecto GNU se inició en los años ochenta con Richard Stallman[10] como máximo responsable, con el fin de crear un sistema operativo totalmente libre.

Actualmente GCC soporta los lenguajes más comunes, como C, C++, Java o Fortran, mientras que de forma no estándar se añaden otros como mercury o VHDL. También soporta multitud de plataformas diferentes además de la x86 clásica de los ordenadores. Por ejemplo, tiene soporte para ARM (tipo de arquitectura muy extendida entre dispositivos móviles como teléfonos o tabletas) o PowerPC (arquitectura que actualmente solo se encuentra en videoconsolas o estaciones para cálculos complejos). Además es el compilador integrado en multitud de famosos entornos de desarrollo multiplataforma, como Eclipse, Netbeans o Code::Blocks.

3.3.2. Doxygen

Figura 11: Logo de Doxygen.

Doxygen es un generador de documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft), VHDL y en cierta medida para PHP, C# y D. Dado que es fácilmente adaptable, funciona en la mayoría de sistemas Unix así como en Windows y Mac OS X. La mayor parte del código de Doxygen está escrita por Dimitri van Heesch[11].

Fue creado en 1997, y está programada en C++, con licencia GPL (General Public License). Esta licencia fue creada como forma para proteger el software libre, con el fin de impedir la apropiación del código y el robo de libertades.

La documentación está escrita en el propio código, y por tanto es relativamente fácil de mantenerla al día. Doxygen puede cruzar documentación de referencia en el código, de modo que el lector de un documento se puede referir fácilmente al código real.

3.3.3. CMake

Figura 12: Logo de CMake

CMake es una herramienta que que se encarga de la generación de código para ayudar a portar código entre distintas plataformas o compiladores. Para ello genera un proyecto siguiendo una serie de instrucciones marcadas por el usuario, facilitando el desarrollo multiplataforma.

Para poder usar CMake correctamente es necesario crear un fichero de nombre CMakeLists.txt[12], que contendrá los parámetros necesarios, como ficheros que forman parte de la compilación, librerías que hay que incluir o el nombre final de la aplicación. CMake lee dicho archivo y genera un proyecto adecuado al Sistema Operativo, compilador elegido y la ubicación de las librerías que van a ser enlazadas. Como extra es posible generar un Makefile específico para algunos entornos de desarrollo concretos, como Visual Studio o Eclipse.

CMake nació como la necesidad de otorgar facilidad para la construcción multiplataforma. Comenzó basándose en un sistema anterior, packer, y fue recibiendo un gran número de añadidos, principalmente de desarrolladores externos que iban adaptando la herramienta a sus necesidades, dando lugar a lo que conocemos hoy día como CMake. Entre otras características, permite analizar dependencias para los lenguajes C, C++, Fortran y Java. También permite el uso de compilaciones paralelas o cruzadas, además de generar ficheros Makefile específicos para algunos entornos de desarrollo.

3.3.4. Boost

Figura 13: Logo de bibliotecas Boost

Boost es un conjunto de bibliotecas de software libre preparadas para extender las capacidades del lenguaje de programación C++. Tienen licencia BSD, que permite que sea utilizada en cualquier tipo de proyectos, ya sean comerciales o no. Proporciona soporte desde bibliotecas de propósito general hasta abstracciones del sistema operativo.

Actualmente Boost está formada por más de 80 bibliotecas individuales, incluidas las bibliotecas de álgebra lineal, la generación de números pseudoaleatorios, multihilos, procesamiento de imágenes, expresiones regulares y pruebas unitarias entre otros. La mayoría de las bibliotecas Boost están basadas en cabeceras, funciones en línea y plantillas, por lo que no tienen que ser construidas antes de su uso. Varios fundadores de Boost pertenecen al Comité ISO de Estándares C++, y además, la próxima versión estándar de C++ incorporará varias de estas bibliotecas.

4. Diseño y desarrollo de la librería SGPS

En informática, una biblioteca o librería es un conjunto de subprogramas que se utilizan para desarrollar otros programas[13]. La característica distintiva es que una biblioteca se organiza con el fin de ser reutilizada por programas independientes, y el usuario sólo necesita conocer la interfaz, y no los detalles internos de la biblioteca.

Los programas se enlazan con las librerías mediante llamadas en el código del mismo, lo que hace que a la hora de compilar dichas librerías se enlacen en el ejecutable. Lo que distingue a la llamada de una biblioteca frente a la de otra función en el mismo programa, es la forma en la que el código está organizado en el sistema: el código de una librería está organizado de tal manera que puede ser utilizado por múltiples programas que no tienen ninguna relación entre sí, mientras que una función de un programa está organizada para ser usada sólo dentro del mismo. Además, ayudan al intercambio de código de forma modular y la facilidad de la distribución del código.

4.1. Librerías de apoyo utilizadas

Para la ejecución de la librería diseñada se ha necesitado un conjunto de otras librerías que han ayudado al buen funcionamiento de la misma.

4.1.1. Boost

Este conjunto de bibliotecas han sido escogidas porque sirven como complemento para STL (Standard Template Library), es decir, proporcionan múltiples funciones que las bibliotecas estándares no ofrecen. Están muy bien documentadas, lo que ayuda en su utilización.

Para el diseño de la librería se ha usado solo algunas de las bibliotecas que boost proporciona. Una de ellas es la biblioteca de *Algorithm String*, extensión de la bilioteca STL que añade funciones de recorte, conversión y reemplazo de funciones, entre otras. Todas ellas vienen en diferentes variantes, lo que aumenta la versatilidad y facilita la elección de la mejor opción para una necesidad particular. Otra librería usada es *Filesystem*, que proporciona facilidades para la instalación de archivos y directorios. En concreto se ha utilizado la función *create_directory* para la creación del directorio de salida en el caso de que la ruta facilitada no exista. La ventaja principal de esta función frente a *mkdir* es que la facilitada por boost se puede utilizar en todos los sistemas operativos. Junto a estas dos, se ha usado otra librería de apoyo, para conocer el tiempo de ejecución. Esta función pertenece a la biblioteca *Progress*.

4.1.2. Gnuplot

Como ya se ha comentado gnuplot es un programa de líneas de comandos para dibujar gráficas en dos y tres dimensiones. Se ha elegido porque es compatible con el lenguaje C++, pudiéndose incluir en un programa mediante la implementación

de alguna biblioteca específica. Para este proyecto se ha utilizado [14].

Dicha librería tiene un problema de múltiple definición si se utiliza dentro de un mismo proyecto en más de una clase, ya que utiliza una definición de función normal en lugar de sólo definición de funciones en línea. Además, algunas variables se duplican en la fase de vinculación, por lo que para su implementación hace falta modificar dicha biblioteca. Dicha modificación viene explicada en la propia página del proyecto: basta con dividir la librería en dos archivos, un .h para la definición de funciones y un .cpp para el comportamiento de estas. Aún con este problema, la simplicidad de implementación y facilidad de utilización de las funciones de esta librería, comparada con otras utilizadas, la ha convertido en la elección idónea.

4.2. Patrones de diseño

Un patrón de diseño es una solución de diseño de software a un problema cuya principal característica es la posibilidad de reutilización. Debe poder utilizarse en diferentes problemas de diseño y en distintas circunstancias. Su finalidad es evitar la reiteración en la búsqueda de un problema ya resuelto.

Los antecedentes de estos patrones datan de 1970 en el campo de la arquitectura. Christopher Alexander escribió varios libros de urbanismo y plantea la posibilidad de reutilizar diseños ya aplicados en construcción. Alexander lo definía así: *Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y describe la esencia de la solución a ese problema, de tal modo que pueda utilizarse esta solución un millón de veces más, sin siquiera hacerlo de la misma manera dos veces*[15].

En las siguientes subsecciones se presentan los patrones utilizados.

4.2.1. Patrón de diseño Singleton

Este patrón se usa en la clase de opciones, en las que se guardan parámetros importantes que sean difíciles de pasar a otras partes del programa, ya que debe ser una clase accesible. La característica principal de este Singleton es que sólo se puede crear una instancia en un programa. Su utilidad radica en que en la librería se utiliza para guardar las configuraciones que se pasarían al programa cuando es ejecutado.

Este patrón de diseño es fácil de utilizar, tan sólo se debe implementar en una clase de la siguiente manera:

```
class Options {

public:
    static Options * Instance();

    static void destroyOptions(){
          if(instance_ != NULL) delete instance_;
        }


protected:
    Options();
    Options (const Options & ) ;
    Options & operator = (const Options & ) ;



private:
    static Options * instance_;
```

Figura 14: Patrón de diseño singleton. Archivo .h

```
Options * Options::instance_ = 0;   // Pointer initialised to 0.

Options * Options::Instance () {
    if (instance_ == 0) {           // First call to this class?
        instance_ = new Options;    // If yes, we instantiate the class.
        atexit(&destroyOptions);
    }

    return instance_;
}


Options::Options() {

}
```

Figura 15: Patrón de diseño singleton. Archivo .cpp

La función *Instance()* comprueba si esta clase se ha instanciado alguna vez para evitar hacerlo de nuevo. Por otro lado, la función *destroyOptions()* se encarga de borrar los datos al final del programa. Esto no es necesario para la funcionalidad de la librería (de hecho, no todos los Singleton disponen de ello) pero evita algunos problemas si mantiene recursos abiertos como canales abierto entre dos programas, llamados *sockets*, o ficheros.

Para la utilización del patrón Singleton en cualquier parte de un proyecto basta con realizar una llamada de la siguiente forma: *Options * opt = Options::Instance()*.

4.2.2. Patrón de diseño Experto en Información

Otro patrón utilizado es el Experto en Información, que es uno de los cinco patrones GARSP (object-oriented design General Responsibility Assignment Soft-

ware Patterns) fundamentales. Los GARSP vienen definidos por un nombre, un problema y una solución.

Este patrón resuelve el problema de la asignación de responsabilidades dentro de la programación orientada a objetos. La solución dada es la de asignar estas responsabilidades al Experto en Información, que en el caso de la librería del presente proyecto es la clase Day. En ella se almacenan todas las variables importantes para el cálculo de las coordenadas terrestres.

4.3. Estilos del diseño

Para la creación de esta biblioteca se ha reutilizado parte del código de otras preexistentes: las funciones *parseArguments()* y *findArguments()*, en las clases SGPS y Console, han sido utilizadas para pasarle ciertos parámetros al programa en el momento de su ejecución. Si, por ejemplo, se pone un '-l' como línea de comandos, la librería entendería que hay que usar la clase Logger, y guardaría los datos necesarios, como archivos de coordenadas o datos del lugar.

Para este tipo de funciones se ha tomado como referencia el método usado por la PCL (Point Cloud Library). Un ejemplo puede verse en las figuras 16 y 17: primero se comprueba si los argumentos pasados por línea de comandos son mayores de 2, ya que el primero es la llamada al ejecutable y el segundo la ruta donde se encuentran los archivos. Después se llama a la función findArguments() de Console (imagen 17) para que esta compruebe si se cumple alguna de estas condiciones (-l, -c, -p...). Si lo hace, se devolverá el número del índice en el que se ha pasado esa opción, y si este es mayor que 1 se activará la opción correspondiente (imagen 16).

```
if (argc > 2){

    int index = Console::parseArguments(argc, argv, "-l", v);

    if (index > 1) { // index > 1 means that this option has been found in the arguments.
        opt-> setLoggerOn();
        Logger::startFiles();
    }

    index = Console::parseArguments(argc, argv, "-c", v);

    if (index > 1)
        opt-> setConsoleOn();

    index = Console::parseArguments(argc, argv, "-p", v);
    if (index > 1)
        opt-> setPlotterOn();

    index = Console::parseArguments(argc, argv, "-w", v);
    if (index > 1)
        opt-> setWorldOn();

    index = Console::parseArguments(argc, argv, "-r", v);
    if (index > 1)
        opt-> setFileReadedOn();
```

Figura 16: Ejemplo función *parseArguments()* en SGPS

```
int Console::parseArguments (int argc, char** argv, const char* str, std::string &val)
{
  int index = findArguments (argc, argv, str);

  if (index > 1  && index < argc)
    val = argv[index];

  return index;
}
```

Figura 17: Ejemplo función *parseArguments()* en Console

Otro estilo de diseño empleado es la forma de estructurar las carpetas. Con el fin de ayudar a la búsqueda y visualización de ficheros para su modificación, se ha decidido separar las clases de los ejemplos, colocando las primeras en la carpeta 'src' y los segundos en la carpeta 'apps'. Por otro lado, las clases también se han separado en dos subcarpetas: 'include', para los .h, y otra también llamada 'src' para los .cpp.

4.4. Librería diseñada

La función de la librería diseñada es la correcta utilización del sistema de geoposicionamiento SGPS, que, como ya se ha explicado anteriormente, indica la ubicación de un objeto a partir de la fecha y los datos de la salida y la puesta de sol de un día determinado. Por tanto, para la creación de la biblioteca se parte de archivos en los que figure la variación de la intensidad de luz durante el transcurso

de un día. La librería se divide en un conjunto de clases entrelazadas, divididas en principales y de apoyo, cuyo diagrama de clases se muestra en la siguiente figura.

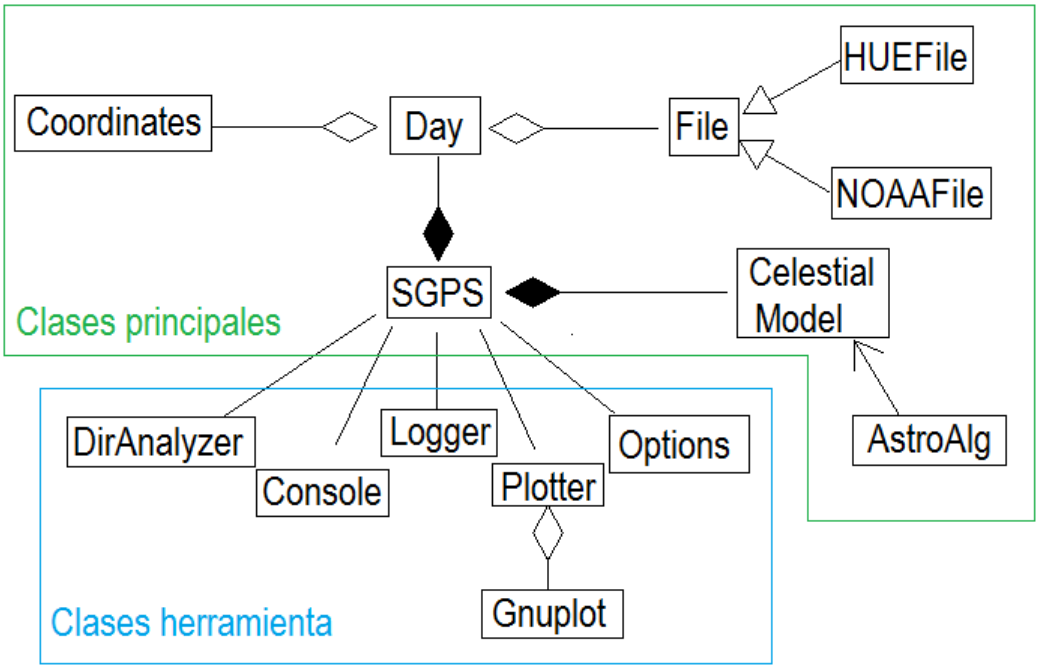

Figura 18: Diagrama clases de la biblioteca diseñada

La librería consta de una clase principal que se ocupa de distribuir los parámetros enviados inicialmente al programa. Una vez hecho esto, las demás clases realizarán las funciones pertinentes, ya sea el cálculo de las coordenadas, la muestra por pantalla o el dibujo de las gráficas.

En las siguientes subsecciones se explican cada una de estas clases, detallando su función y los problemas que han surgido al implementarlas. La biblioteca diseñada está disponible en [16], junto con algunos ejemplos de uso y el archivo que genera la documentación, donde se encuentra la explicación de todas las demás funciones.

4.4.1. Clase SGPS

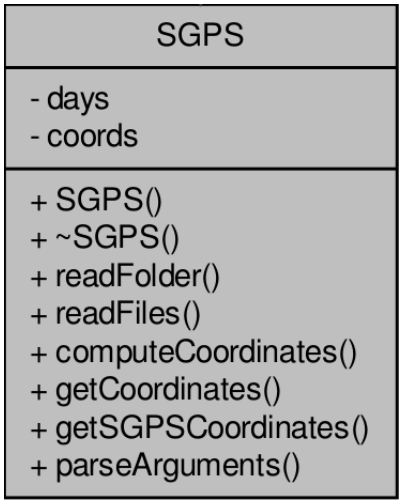

Figura 19: Diagrama UML de sgps.h

SGPS es la clase principal, de la que parten todas las demás, y su función es distribuir al resto de clases las variables introducidas en el programa: envía la ruta de los archivos a dirAnalyzer para su análisis y los argumentos pasados por línea de comando primero a Console, para comprobar las opciones, y luego a Options, para ajustarlas. También se encarga de mandar la información de los días a Celestial Model, para que este calcule las coordenadas. Las funciones de SGPS son:

- *readFolder()*. Es la encargada de pasar la ruta de los archivos a analizar a la clase dirAnalyzer, para que esta separe los archivos válidos de los que no lo son.
- *readFiles()*. Envía la ruta de los archivos correctos a dirAnalyzer para obtener la información de los días, que se guardará en un vector.
- *computeCoordinates()*. Recorre el vector de días correctos analizados en dirAnalyzer, mandando dicha información a Celestial Model, donde se calcularán las coordenadas. Da opción a mostrar la información por consola, crear un archivo con ella o dibujar una gráfica con las coordenadas calculadas.
- *parseArguments()*. Activa las opciones pasadas, previamente comprobadas en Console.
- *getCoordinates()* y *getSGPSCoordinates()*. Devuelven el número total de coordenadas reales calculadas en los programas, y un vector con sus valores. Son útiles para saber si el número de coordenadas calculadas es igual al de archivos analizados.

4.4.2. Clase DirAnalyzer

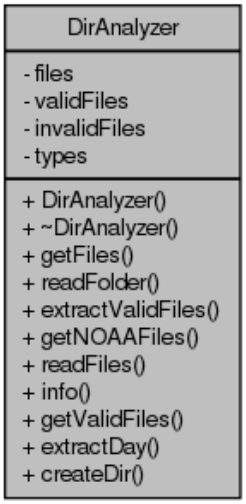

Figura 20: Diagrama UML de diranalyzer.h

Realiza todas las funciones relacionadas con los directorios o archivos del proyecto, proporcionando abstracción de los tipos de archivos. Es una de las clases de apoyo, ya que facilita la labor de SGPS pero su uso no es obligatorio, pudiendo ser sustituida por otro método de extracción de datos.

Para las labores de análisis, DirAnalyzer entra en las carpetas y subcarpetas del proyecto, guardando en un vector las rutas de todos y cada uno de los archivos del mismo. A continuación, comprueba uno por uno la validez de estos utilizando diversos métodos, entre ellos la lectura del nombre del archivo, la comprobación del tamaño o la lectura de su primera línea. Por último, instancia un objeto de la clase File (o derivadas) para extraer la información de los archivos válidos. Existe un método manual para facilitar el trabajo de la clase, indicando el tipo de archivo a leer. Para ello, al ejecutar el programa se pasan por línea de comandos dos parámetro: *-f* y el tipo de archivo (*generic, noaa o huefile*). DirAnalyzer también crea una carpeta para que Logger guarde los archivos generados y manda el vector con todos los archivos para que sean mostrados por Console.

4.4.3. Clase File

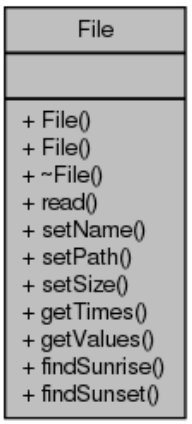

Figura 21: Diagrama UML de file.h

File es la encargada de extraer la información de cada uno de los ficheros que vienen de DirAnalyzer. De File heredan dos clases: NOAAFile y HUEFile.

File

Es la encargada de leer archivos genéricos. Para comprobar que los ficheros son de este tipo lee su primera línea, en la que debe encontrar *genericfile*. Los archivos genéricos solo tienen la información del lugar (coordenadas reales, día, mes y año) y los datos de iluminación de todo el día junto a la hora a la que se ha tomado, como se puede ver en la figura 22

```
genericfile
-3.9282 40.313769
11      2       2013
00:00:12        0
00:00:42        0
00:01:12        0
00:01:42        0
00:02:12        0
00:02:42        0
00:03:12        0
00:03:42        0
00:04:12        0
00:04:42        0
00:05:12        0
00:05:42        0
00:06:12        0
00:06:42        0
00:07:12        0
00:07:42        0
00:08:12        0
00:08:42        0
00:09:12        0
00:09:42        0
```

Figura 22: Generic file

Uno de los problemas que han surgido al implementar esta clase ha sido encontrar la hora de salida y puesta del sol. La solución escogida ha sido comprobar

uno a uno todos los valores de iluminación hasta que alguno sobrepase cierto valor, manteniéndose en las siguientes medidas; ese momento se tomará como hora de salida del sol. Empleando el mismo método comenzando por los últimos valores se obtendrá la hora de puesta del sol. Una vez hecho esto, devuelve todas las variables obtenidas en un objeto de la clase Day, para que SGPS las guarde en un vector. Esta técnica puede ocasionar algunos errores en días fraccionados, por lo que en el futuro se intentará buscar una forma más adecuada. Otro problema ha sido debido a que en estos archivos la hora viene dada como una cadena de string del tipo *hh:mm:ss*, que ha habido que traducir al utilizado por SGPS (de 0 a 24 horas). Para realizar este cambio se han separado las horas, los minutos y los segundos del string mediante la función *atoi()*, que convierte de 'char' a 'int'.

NOAAFile

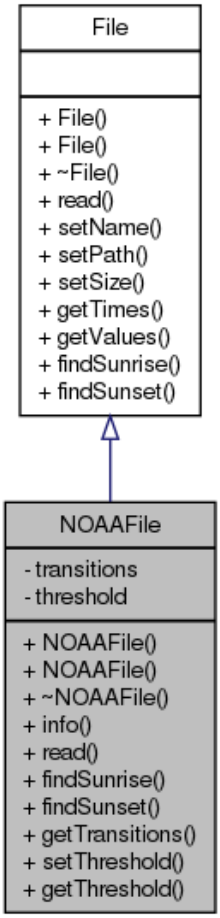

Figura 23: Diagrama UML de noaafile.h

Esta clase hereda gran parte de las funciones de file, pero como se puede ver en la figura 23 se necesita una forma especifica tanto en la lectura de sus archivos como en la forma de encontrar su salida y puesta de sol. Los ficheros NOAAFile, caracterizados por tener un tamaño dentro de un rango determinado, son facilitados por el National Oceanic and Atmospheric Administration y no sólo muestran históricos de iluminación, por lo que durante su lectura es preciso seleccionar los datos necesario y desechar los demás. Además, no todos los archivos

del NOAA son válidos, ya que algunos han sido tomados en equinoccios. Para su lectura se toma la primera línea, en la que estará la información del lugar. La hora se extrae de las columnas 4, 5 y 6, y la iluminación, junto con una indicación de la fiabilidad, se encuentran en las columnas 15 y 16. La estructura se puede ver en la siguiente figura:

```
Bondville
  40.05  -88.37  213 m version 1
2013   8  1  8  0  0  0.000 104.00    -4.6 0    -0.2 0     0.4 0    -0.7 0   225.5 0   273.3 0   272.9 0   302.8 0   272.5 0   272.5 0     0.0 0
2013   8  1  8  0  1  0.017 104.18    -4.6 0    -0.2 0     0.4 0    -0.7 0   225.7 0   273.3 0   272.9 0   302.9 0   272.5 0   272.5 0     0.0 0
2013   8  1  8  0  2  0.033 104.36    -4.7 0    -0.2 0     0.4 0    -0.7 0   226.1 0   273.3 0   272.9 0   303.0 0   272.5 0   272.5 0     0.0 0
2013   8  1  8  0  3  0.050 104.54    -4.7 0    -0.2 0     0.4 0    -0.7 0   226.5 0   273.3 0   272.9 0   303.2 0   272.5 0   272.4 0     0.0 0
2013   8  1  8  0  4  0.067 104.73    -4.7 0    -0.2 0     0.5 0    -0.7 0   226.8 0   273.3 0   272.9 0   303.2 0   272.5 0   272.4 0     0.0 0
2013   8  1  8  0  5  0.083 104.91    -4.7 0    -0.2 0     0.4 0    -0.7 0   227.2 0   273.3 0   272.9 0   303.3 0   272.5 0   272.4 0     0.0 0
2013   8  1  8  0  6  0.100 105.09    -4.8 0    -0.2 0     0.4 0    -0.7 0   227.5 0   273.3 0   272.9 0   303.6 0   272.5 0   272.4 0     0.0 0
2013   8  1  8  0  7  0.117 105.28    -4.8 0    -0.2 0     0.4 0    -0.7 0   227.9 0   273.3 0   272.9 0   303.6 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0  8  0.133 105.46    -4.8 0    -0.1 0     0.3 0    -0.7 0   228.2 0   273.3 0   272.9 0   303.8 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0  9  0.150 105.64    -4.8 0    -0.2 0     0.3 0    -0.6 0   228.6 0   273.3 0   272.9 0   304.0 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0 10  0.167 105.83    -4.8 0    -0.2 0     0.4 0    -0.6 0   228.9 0   273.3 0   272.9 0   304.2 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0 11  0.183 106.01    -4.9 0    -0.2 0     0.4 0    -0.6 0   229.3 0   273.3 0   272.9 0   304.4 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0 12  0.200 106.20    -5.0 0    -0.2 0     0.4 0    -0.6 0   229.6 0   273.3 0   272.9 0   304.1 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0 13  0.217 106.38    -4.9 0    -0.2 0     0.3 0    -0.6 0   229.9 0   273.3 0   272.9 0   304.0 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0 14  0.233 106.56    -4.9 0    -0.1 0     0.4 0    -0.5 0   230.2 0   273.3 0   272.9 0   304.1 0   272.4 0   272.4 0     0.0 0
2013   8  1  8  0 15  0.250 106.75    -4.9 0    -0.1 0     0.3 0    -0.5 0   230.5 0   273.3 0   272.9 0   304.3 0   272.4 0   272.3 0     0.0 0
2013   8  1  8  0 16  0.267 106.93    -4.9 0    -0.1 0     0.3 0    -0.5 0   230.9 0   273.3 0   272.9 0   304.4 0   272.4 0   272.3 0     0.0 0
2013   8  1  8  0 17  0.283 107.12    -4.9 0    -0.1 0     0.4 0    -0.4 0   231.1 0   273.2 0   272.9 0   304.6 0   272.4 0   272.3 0     0.0 0
2013   8  1  8  0 18  0.300 107.30    -4.9 0    -0.2 0     0.4 0    -0.3 0   231.4 0   273.2 0   272.9 0   304.7 0   272.4 0   272.3 0     0.0 0
2013   8  1  8  0 19  0.317 107.49    -4.9 0    -0.1 0     0.3 0    -0.3 0   231.6 0   273.3 0   272.9 0   304.6 0   272.4 0   272.3 0     0.0 0
2013   8  1  8  0 20  0.333 107.67    -4.9 0    -0.1 0     0.2 0    -0.2 0   231.9 0   273.3 0   272.9 0   304.8 0   272.3 0   272.3 0     0.0 0
2013   8  1  8  0 21  0.350 107.86    -5.0 0    -0.1 0     0.3 0    -0.2 0   232.1 0   273.2 0   272.9 0   304.8 0   272.3 0   272.3 0     0.0 0
2013   8  1  8  0 22  0.367 108.04    -4.9 0    -0.1 0     0.3 0    -0.2 0   232.3 0   273.2 0   272.9 0   305.0 0   272.3 0   272.3 0     0.0 0
2013   8  1  8  0 23  0.383 108.23    -5.1 0    -0.1 0     0.3 0    -0.1 0   232.7 0   273.2 0   272.9 0   305.3 0   272.3 0   272.3 0     0.0 0
2013   8  1  8  0 24  0.400 108.41    -5.0 0    -0.2 0     0.4 0    -0.1 0   233.1 0   273.3 0   272.9 0   305.3 0   272.3 0   272.3 0     0.0 0
```

Figura 24: Archivo NOAA

HUEFile

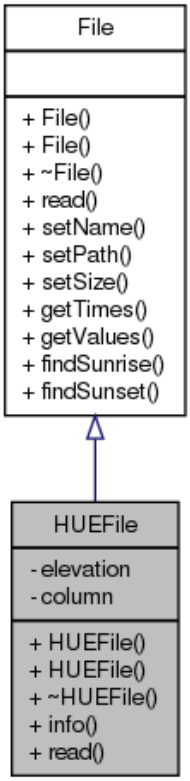

Figura 25: Diagrama UML de huefile.h

Esta clase hereda todo de File excepto la forma de leer y las variables 'column' y 'elevation'. Cuando se leen estos archivos es necesario facilitarle el valor de la

columna, ya que dispone de varias magnitudes y la forma de leer y procesar los datos cambia en función de ellas. HUEFile se explicará en la sección 6.1, ya que para entender su funcionamiento se tiene que explicar una de la ampliaciones de la librería. Su estructura es la siguiente:

3.77	40.23	Casa_Jose		2	4	2013	648						
0	0	24	1.01	0.86	0.01	0.01	0.00	0.99	1.02	0.98	3.59	-0.19	0.18
0	1	28	1.01	0.79	0.01	0.02	0.00	0.99	1.02	0.98	3.59	-0.18	0.17
0	2	31	1.01	0.79	0.01	0.01	0.00	0.99	1.03	0.98	3.59	-0.19	0.17
0	3	34	1.01	0.79	0.01	0.01	0.00	0.99	1.02	0.98	3.59	-0.18	0.18
0	4	38	1.01	0.79	0.01	0.01	0.00	0.99	1.02	0.98	3.59	-0.19	0.18
0	5	41	1.01	0.83	0.01	0.02	0.00	0.99	1.03	0.98	3.59	-0.18	0.17
0	6	45	1.01	0.86	0.01	0.02	0.00	0.99	1.02	0.98	3.59	-0.19	0.17
0	7	48	1.01	0.90	0.01	0.01	0.00	0.99	1.03	0.98	3.59	-0.18	0.17
0	8	52	1.01	0.86	0.01	0.01	0.00	0.99	1.03	0.98	3.59	-0.19	0.17
0	9	55	1.01	0.75	0.01	0.02	0.00	0.99	1.03	0.98	3.59	-0.18	0.17
0	10	59	1.01	0.86	0.01	0.02	0.00	0.99	1.02	0.98	3.59	-0.18	0.18
0	12	2	1.01	0.83	0.01	0.02	0.00	0.99	1.03	0.98	3.59	-0.18	0.17
0	13	6	1.01	0.75	0.01	0.01	0.00	0.99	1.02	0.98	3.59	-0.19	0.18
0	14	9	1.01	0.86	0.01	0.02	0.00	0.99	1.02	0.98	3.59	-0.18	0.17
0	15	13	1.01	0.83	0.01	0.01	0.00	0.99	1.03	0.98	3.59	-0.19	0.17
0	16	17	1.01	0.90	0.01	0.01	0.00	0.99	1.03	0.98	3.59	-0.19	0.18
0	17	20	1.01	0.75	0.01	0.02	0.00	0.99	1.02	0.98	3.59	-0.18	0.17
0	18	24	1.01	0.75	0.01	0.01	0.00	0.99	1.02	0.98	3.59	-0.19	0.17
0	19	28	1.01	0.83	0.01	0.01	0.00	0.99	1.02	0.98	3.59	-0.19	0.18
0	20	31	1.01	0.75	0.01	0.02	0.00	0.99	1.02	0.98	3.59	-0.18	0.17
0	21	35	1.01	0.90	0.01	0.02	0.00	0.99	1.03	0.98	3.59	-0.18	0.17

Figura 26: Archivo HUE

4.4.4. Clase Day

Figura 27: Diagrama UML de day.h

Day almacena las variables que salen de los ficheros examinados en File y prácticamente todas sus funciones son para guardar o enviar esas variables a otras partes del programa. Para su diseño se han utilizado los métodos *get* y *set*, que sirven para la visualización o modificación de variables privadas de una clase. Las variables se hacen privadas para que únicamente puedan ser modificadas desde dentro de la propia clase, impidiendo así a las demás su uso. Además de las funciones *get* y *set* existe *plotDay()*, que sirve como enlace entre la clase que la llama y la clase que realiza las gráficas.

4.4.5. Clase CelestialModel

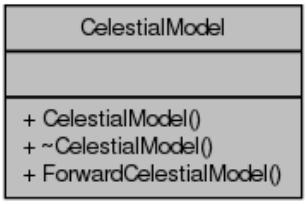

Figura 28: Diagrama UML de celestialmodel.h

CelestialModel se encarga de calcular las coordenadas y de enviar el vector con la información de los días que vienen de SGPS hacia AstroAlg para que realice los cálculos. Utiliza el modelo descrito en la sección 2.2.2 y al estar creado únicamente para SGPS se ha decidido que vaya en una clase separada de las demás.

4.4.6. Clase AstroAlg

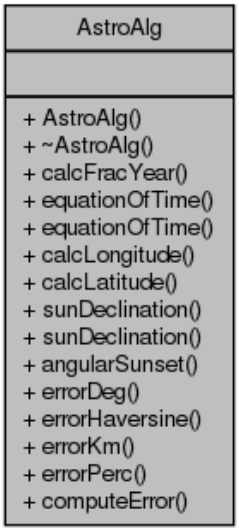

Figura 29: Diagrama UML de astroalg.h

Esta clase es la encargada de hacer todos los cálculos matemáticos, ya sea para conocer la longitud, la latitud o el error cometido. Para el cálculo de la longitud y la latitud utiliza el modelo explicado anteriormente, y con respecto a los errores

hace tres cálculos distintos: el error en grados, que es la resta de las coordenadas reales respecto a las calculadas, el error en porcentaje, que es el error en tanto por ciento respecto del total de las coordenadas reales y el error en kilómetros. Este último se calcula de dos formas diferentes: la primera mediante el método *Haversine*, calculando el error en kilómetros en línea recta, y la segunda hallando el error en kilómetros de la longitud y la latitud por separado.

4.4.7. Clase Console

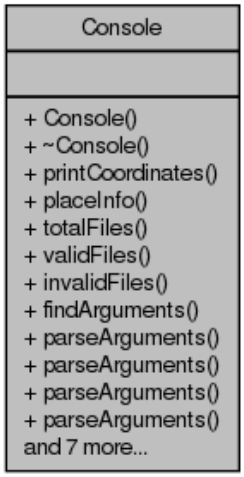

Figura 30: Diagrama UML de console.h

La clase Console tiene dos funcionalidades: Se encarga de todos los mensajes que salen por pantalla, como la muestra e información de los archivos válidos e inválidos o los mensajes de error, y también de las iteraciones del usuario mediante consola. También, como ya se ha explicado, contiene las funciones *findArgumets()* y *parseArgumets()* que comprueban si las opciones pasadas por línea de comando son válidas.

4.4.8. Clase Logger

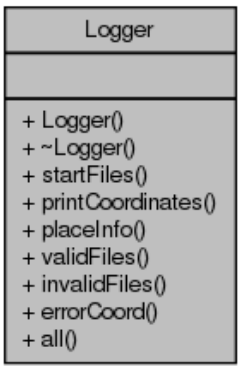

Figura 31: Diagrama UML de logger.h

Logger se encarga de guardar toda la información en archivos para su visualización una vez finalizado el programa. Crea 7 tipos de archivo:

- *Valid Files*. Este fichero contiene el nombre del archivo, la ruta del mismo, y la información básica que contiene: lugar, coordenadas reales, día y año.
- *Invalid Files*. Contiene el nombre y la ruta de los archivos inválidos.
- *Place Information*. En él se almacena la información extraída de los ficheros (lugar, coordenadas, día y año) y las horas de salida y puesta del sol.
- *Error Coordinates*. Este fichero ayuda a la comparación de resultados, ya que muestra las coordenadas reales y calculadas junto con todos los errores computados por AstroAlg.
- *All Files*. Este archivo contiene toda la información anterior unida.
- *Histogram latitude* e *Histogram longitude*. Estos archivos de tipo .dat almacenan las veces que se ha repetido un rango de errores en el calculo de las coordenadas, para que la clase Plotter pueda graficarlos.

4.4.9. Clase Plotter

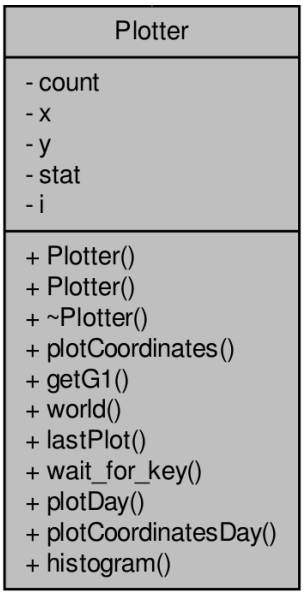

Figura 32: Diagrama UML de plotter.h

Esta clase, junto a la descargada para la utilización de gnuplot, permite que la librería dibuje gráficas, a saber:

- Las coordenadas calculadas por la librería. Puede seleccionarse la opción de que la librería dibuje también un contorno de la Tierra para observar las coordenadas de una forma óptima.
- La iluminación durante un día completo.

- La comparación de las coordenadas reales con las calculadas en un día.
- Un histograma con el promedio de errores cometidos en el cálculo de las coordenadas.

En el diseño de esta clase hubo que tener en cuenta un problema: al dibujar la gráfica esta se cerraba al terminar de ilustrar lo deseado. Para solucionarlo se tuvo que hacer uso de una función adicional llamada *wait_for_key()*, que mantiene el programa inmóvil hasta que se pulse una tecla. De este modo, la gráfica queda en pantalla hasta que se indica que puede cerrarse.

4.4.10. Clase Options

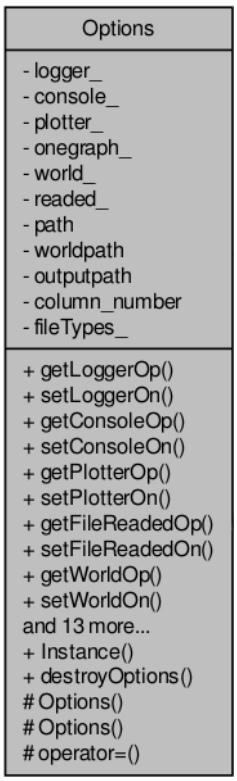

Figura 33: Diagrama UML de options.h

Como ya se ha comentado, Options sigue el patrón de diseño Singleton, y su función es la de activar ciertas partes del programa, a saber:

- *-c*, que activa la visualización por consola.
- *-l*, que activa la opción para guardar ficheros con los datos obtenidos.
- *-p*, que activa la clase Plotter, con lo que aparecerán las gráficas que se necesiten.
- *-g*, que hace que solo se dibuje una gráfica para todos los archivos analizados; en su ausencia, se dibujarían varias gráficas.

- *-w*, que hace que aparezca el contorno del mundo en la gráfica al dibujarse las coordenadas.

Además de esto, Options guarda variables que se necesitan en varias partes del programa y que no están enlazadas, como por ejemplo la ruta de salida de los archivos o la columna que se debe leer para los archivos HUEFile.

5. Ejemplos de utilización de librería

Para comprobar el buen funcionamiento del diseño descrito, se han realizado tres programas que realizan distintas funciones de la librería. A continuación se explicará el funcionamiento de cada uno de ellos junto con un diagrama de secuencia.

5.1. SGPS Test

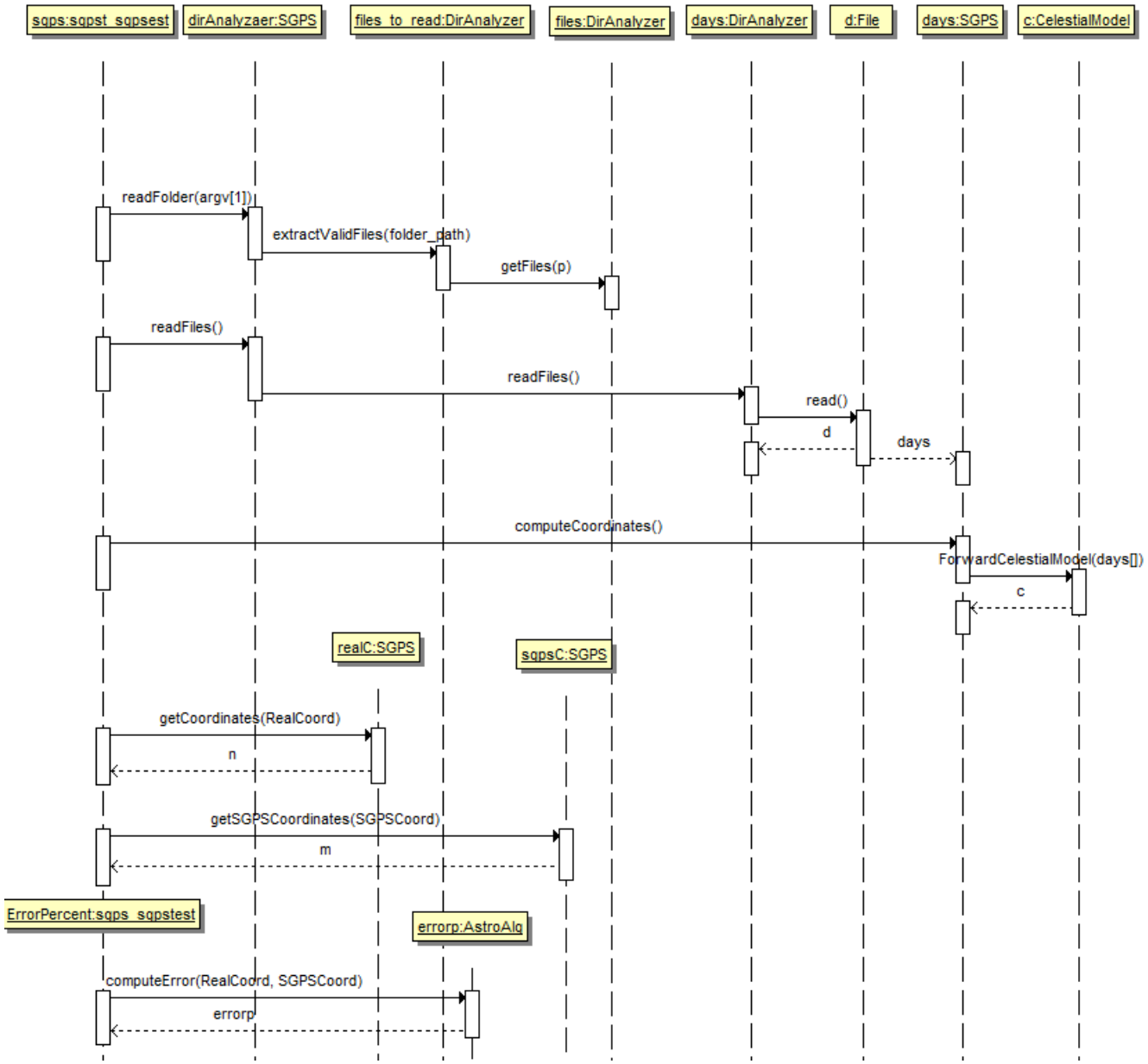

Figura 34: Diagrama secuencia sgpstest

Este primer ejemplo es el más extenso y el que más partes de la biblioteca utiliza. A la hora de ejecutar el programa siempre se le pasará un argumento

por línea de comando que sea un archivo o carpeta. Además se le podrán pasar alguna de las opciones descritas en la sección anterior (-l, -p, -c, etc.). En el caso de ponerle solamente la ruta de archivos se activarán por defecto las funciones de Logger, Console y Plotter. El ejemplo será el mostrado en la figura 35.

```
#include <iostream>
#include <vector>
#include <boost/progress.hpp>

#include "../../base/include/sgps.h"
#include "../../base/include/astroalg.h"
#include "../../base/include/plotter.h"

using namespace std;

int main (int argc, char* argv[]) {
»
»
»       SGPS sgps;
»
»       sgps.parseArguments (argc, argv);
»
»       boost::progress_timer t;

»       sgps.readFolder(argv[1]);
»       sgps.readFiles();
»       sgps.computeCoordinates();
»
»       vector<Coordinates> realCoord;
»       int n = sgps.getCoordinates(realCoord);
»
»       vector<Coordinates> sgpsCoord;
»       int m = sgps.getSGPSCoordinates(sgpsCoord);
»
»       vector<Coordinates> errorPercent = AstroAlg::computeError(realCoord,sgpsCoord);
»
»
»       Options * opt =  Options::Instance();
»
»       if (opt->getPlotterOp() == true){
»       »       Plotter::histogram (errorPercent);
»       }
»
»       return 0;
}
```

Figura 35: sgps_test

Para la explicación de este ejemplo se utilizará siempre esta carpeta con los siguientes trece archivos que pueden verse en la figura 36: dos archivos inválidos, el archivo que dibujará el contorno de la Tierra, ocho archivos .dat que son los facilitados por el NOAA, un archivo genérico y otro archivo realizado por una de las ampliaciones.

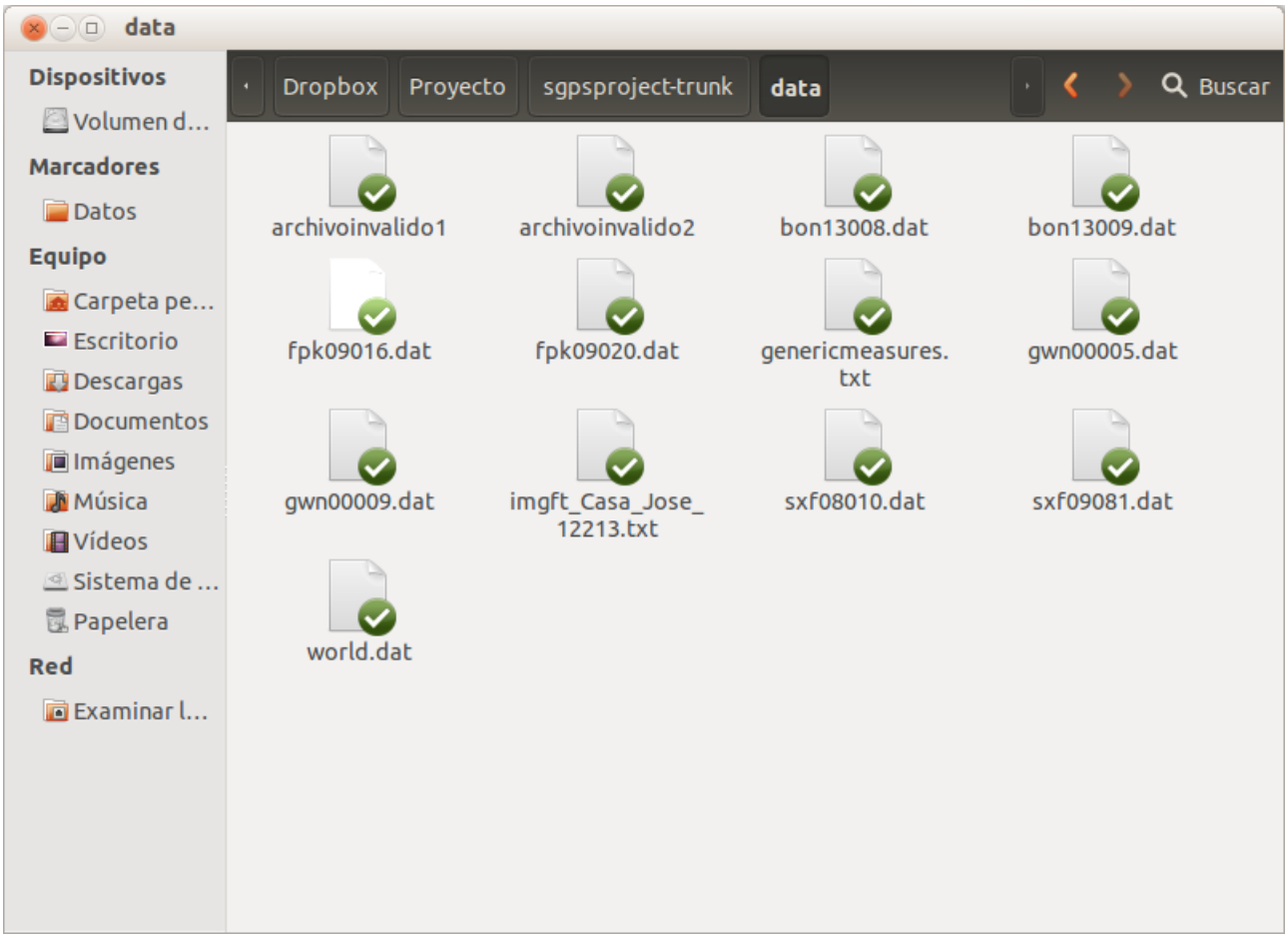

Figura 36: Carpeta con los archivos utilizados de ejemplo

Una vez se ha compilado la librería junto con el programa se ejecuta desde el terminal de la forma que se observa en la figura 37. Como se puede ver se manda también la carpeta descrita anteriormente. Además al no haberle puesto más argumentos en la línea de comandos deberá guardar archivos, mostrar todo por pantalla y realizar las gráficas pertinentes.

```
isaac@isaac-K55VD:~/Dropbox/Proyecto/sgpsproject-trunk/SGPS_CPP/build/bin$ ./sgps_sgpstest
../../../data
```

Figura 37: Forma de ejecutar sgpstest

Una vez se ejecuta el programa lo primero que se hará será crear un objeto de SGPS para poder llamar a las demás funciones. Después se comprobarán las opciones pasadas con la función *parseArguments()* de SGPS. En este caso al no haber nada utilizará las opciones por defecto. Una vez realizada esta comprobación también se mirará el tiempo que tarda en realizar todo el programa con la función *timer* de las bibliotecas boost.

La primera llamada será la función *readFolder(argv[1])* junto con la ruta de los

archivos, que es el segundo argumento (en programación C++ el primer argumento siempre es 0, en este caso la llamada al programa). Esta función se encargará de pasar por todos los archivos de la carpeta y guardará cada una de las rutas en un vector. Una vez que se tienen todos los archivos guardados, se llama a la función *readFiles()* que sacará todos los archivos válidos y guardará toda su información en un vector de la clase Day. El mensaje mostrado por pantalla para estas funciones será el de la figura 38. Como se puede observar descarta los dos archivos inválidos y el fichero world.dat, y se queda con los diez válidos.

```
Output saved in: /home/isaac/Escritorio/output/
Reading File  ../../../data/archivoinvalido1 is invalid, bad extension
Reading File  ../../../data/archivoinvalido2 is invalid, bad extension
Reading File  ../../../data/world.dat is invalid, bad extension
 There are 13 files.
10 are valid files.
File reading ../../../data/bon13008.dat valid
 Bondville 40.05 -88.37 day info: 8 2013 Measures taken: 1440
Day info: 8 2013. Transitions: 2. Sunrise: 13.2915 Sunset: 22.7585
File reading ../../../data/bon13009.dat valid
 Bondville 40.05 -88.37 day info: 8 2013 Measures taken: 1440
Day info: 8 2013. Transitions: 2. Sunrise: 13.2915 Sunset: 22.7585
File reading ../../../data/fpk09016.dat valid
 Fort Peck 48.31 -105.1 day info: 16 2009 Measures taken: 1440
Day info: 16 2009. Transitions: 2. Sunrise: 14.7915 Sunset: 23.575
File reading ../../../data/fpk09020.dat valid
 Fort Peck 48.31 -105.1 day info: 20 2009 Measures taken: 1440
Day info: 20 2009. Transitions: 4. Sunrise: 14.717 Sunset: 23.7165
File reading ../../../data/genericmeasures.txt valid
generic station 40.3138 -3.9282 day info: 42 2013 Measures taken: 2879
Day info: 42 2013. Transitions: 0. Sunrise: 7.26639 Sunset: 17.8231
File reading ../../../data/gwn00005.dat valid
 Goodwin Creek 34.25 -89.87 day info: 5 2000 Measures taken: 480
Day info: 5 2000. Transitions: 2. Sunrise: 13.025 Sunset: 23.175
File reading ../../../data/gwn00009.dat valid
 Goodwin Creek 34.25 -89.87 day info: 9 2000 Measures taken: 480
Day info: 9 2000. Transitions: 2. Sunrise: 13.575 Sunset: 23.175
File reading ../../../data/imgft_Casa_Jose_12213.txt valid
Casa_Jose 40.23 -3.77 day info: 122 2013 Measures taken: 905
Day info: 122 2013. Transitions: 0. Sunrise: 5.54028 Sunset: 19.1225
File reading ../../../data/sxf08010.dat valid
 Sioux Falls 43.73 -96.62 day info: 10 2008 Measures taken: 480
Day info: 10 2008. Transitions: 2. Sunrise: 14.175 Sunset: 22.975
Reading File  ../../../data/sxf09081.dat is invalid, equinox
```

Figura 38: Lectura de archivos

También se puede ver como al ir leyendo todos los archivos y extrayendo la información el último de ellos manda un mensaje de error porque ha sido tomado en un equinoccio, lo que provoca que la medida no sea del todo fiable, por lo que se descartará. En un equinoccio la declinación está muy cerca de cero, lo que puede provocar que en el cálculo de la latitud se obtengan divisiones que tiendan a infinito.

Con toda la información guardada, se pasará al cálculo de las coordenadas, las

cuales se mostraran por pantalla y en una gráfica de la siguiente manera:

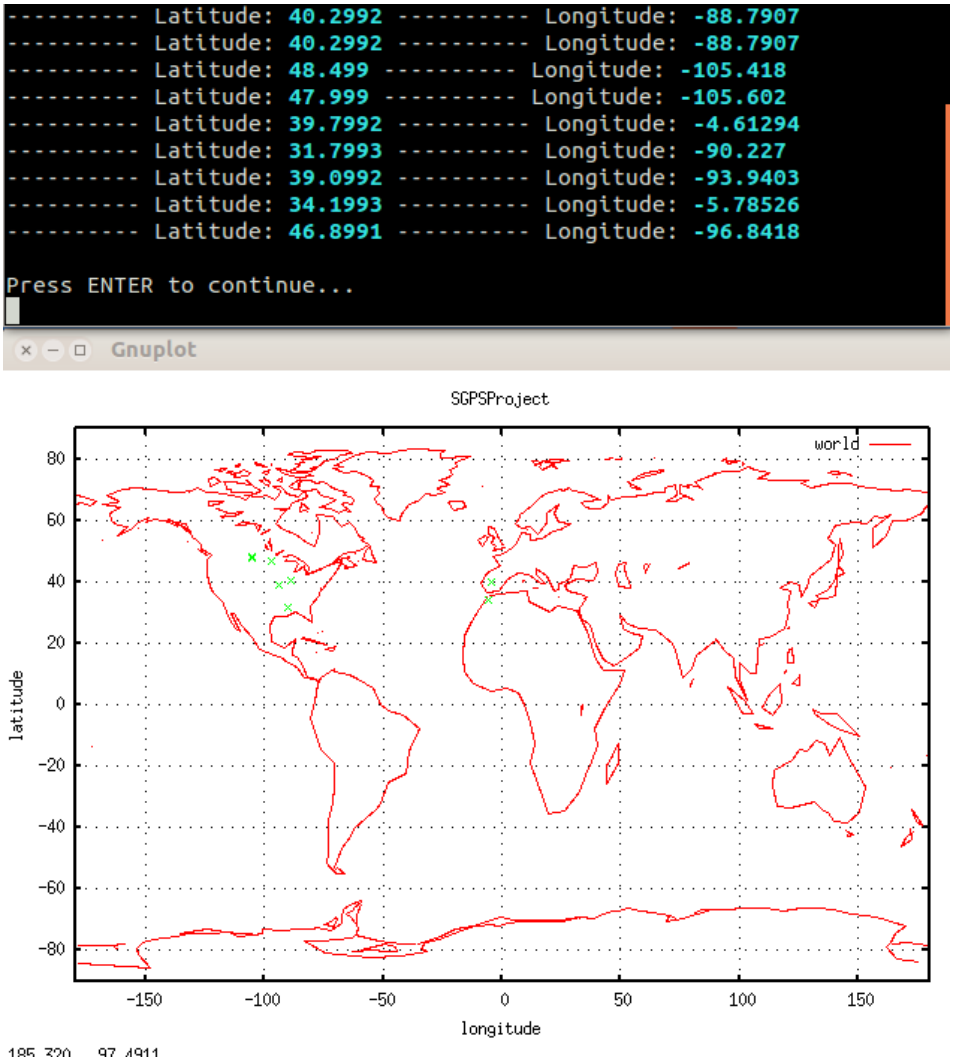

Figura 39: Coordenadas calculadas

Como se observa en la gráfica no queda muy claro a qué coordenadas corresponde cada punto. Esto es debido a un problema con la biblioteca de gnuplot utilizada, que se solucionaría si se creara una librería propia de gnuplot. Esto puede ser uno de los trabajos futuros a realizar.

Cuando se han calculado las coordenadas de todos los archivos, se llama a las funciones *getCoordinates()* y *getSGPSCoordinates()* para guardarlas en vectores y poder mandarlas a AstroAlg para calcular los errores, de los cuales saldrán los archivos de Logger, junto con los histogramas que se ven en la figura 40.

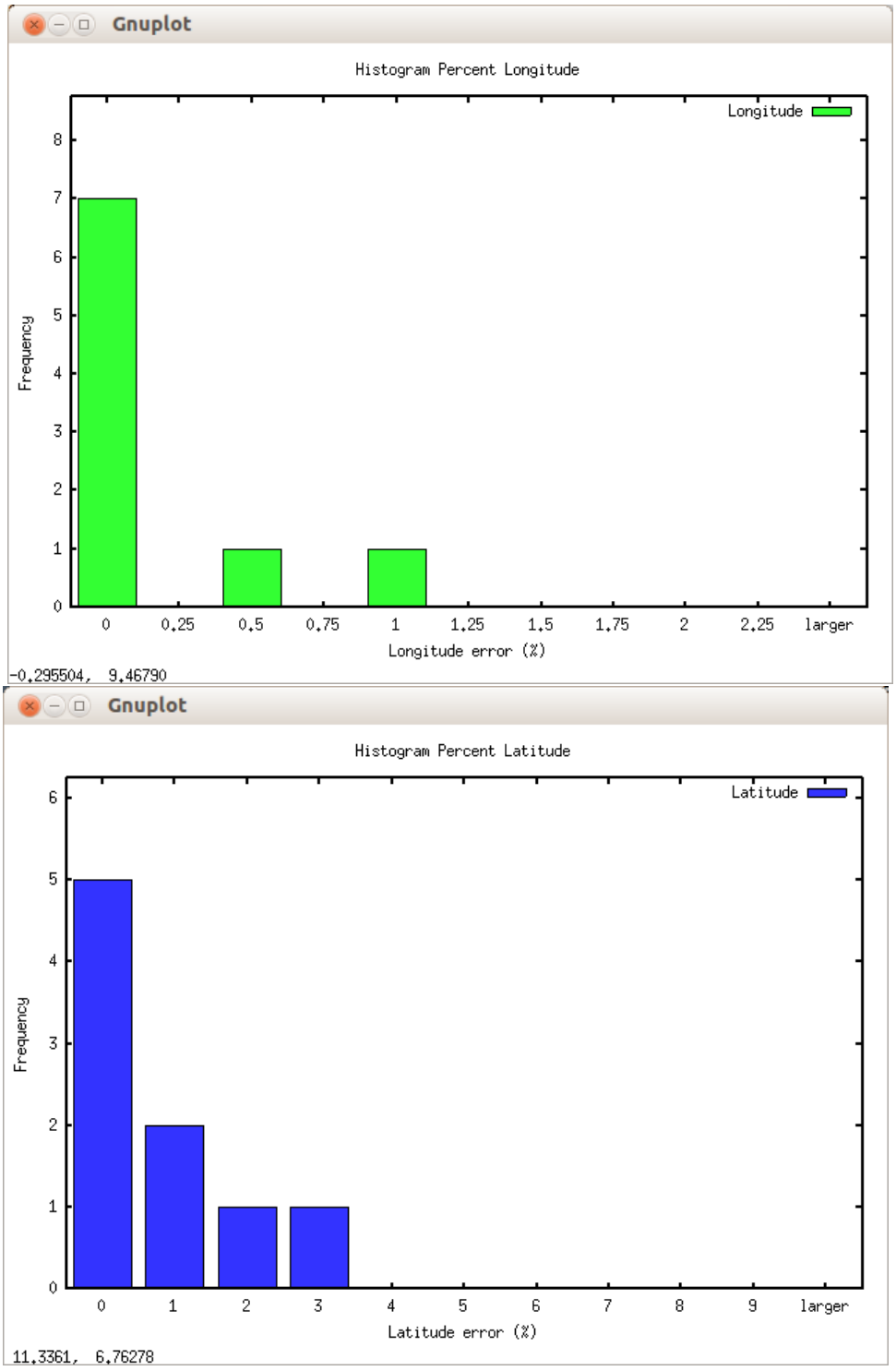

Figura 40: Histogramas de errores

Como se ve en la gráfica de longitud, de los nueve archivos válidos, en siete de ellos se comete un error menor al 0,25 % y tan solo en uno de ellos el error es mayor al 1 %. En la gráfica de latitud cinco de los archivos tienen un error menor al 1 %, siendo el error máximo entre un 3 y un 4 %.

Los ficheros creados en los que se almacenan todos los datos podrán encontrarse

en una carpeta llamada output, creada por defecto en el escritorio, aunque esta ubicación puede cambiarse.

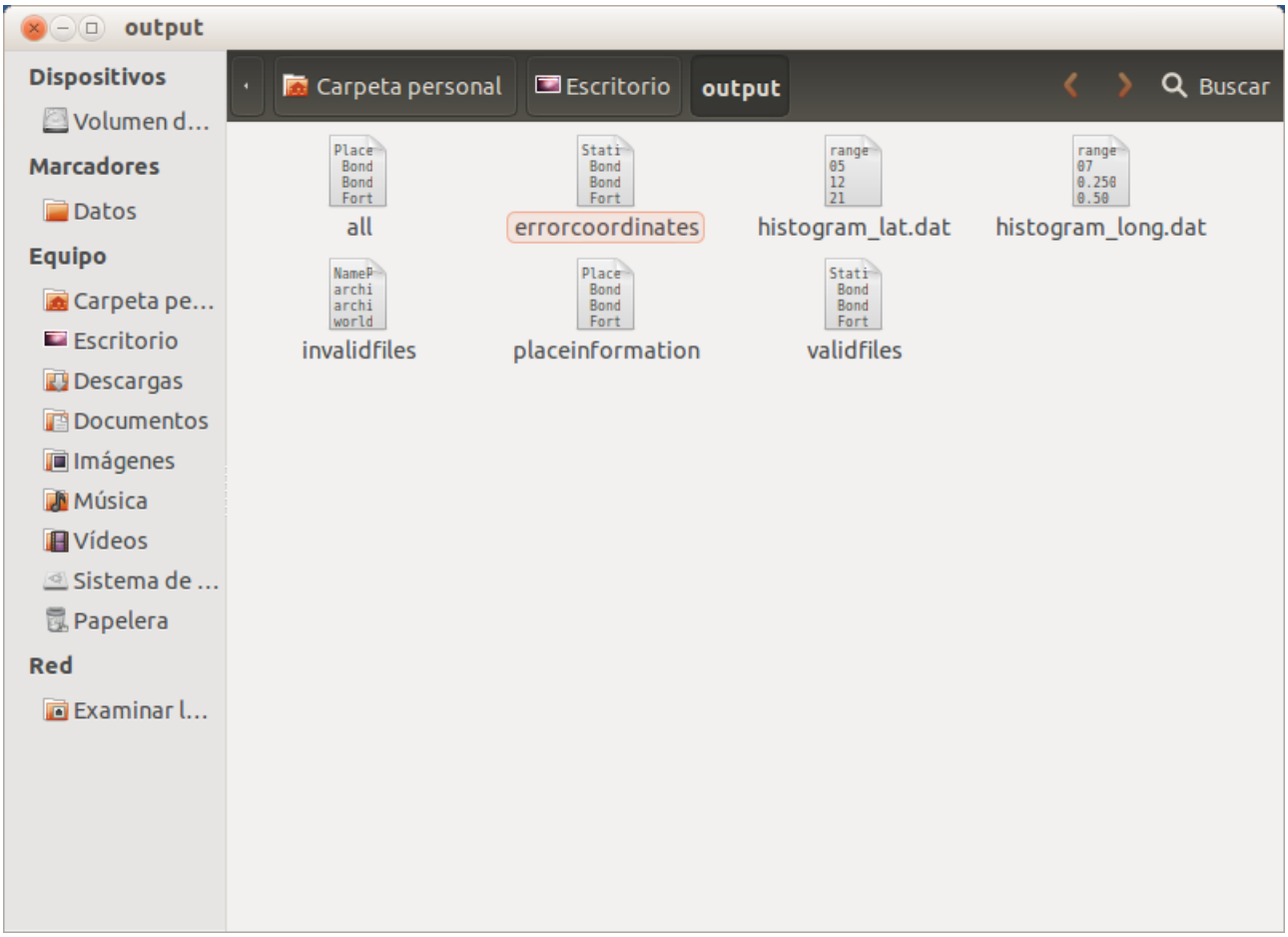

Figura 41: Archivos de salida

5.2. Plot Day

El diagrama de secuencia de este ejemplo puede verse en a figura 42.

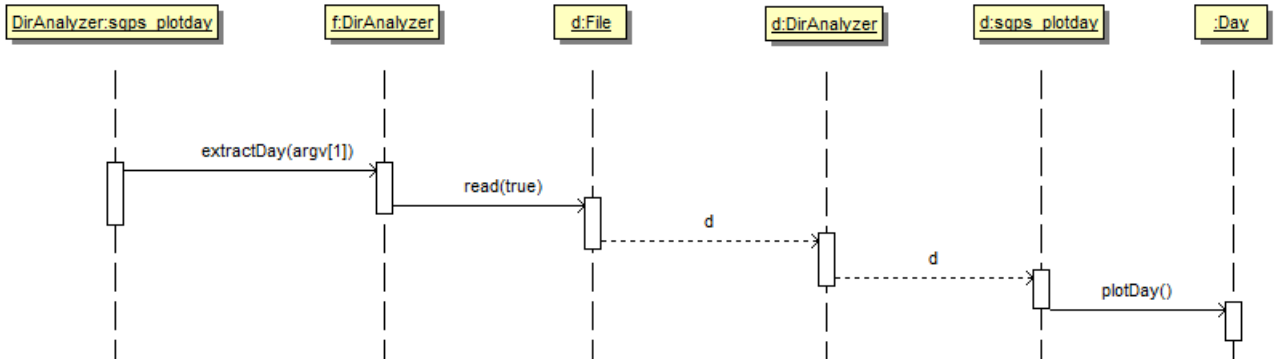

Figura 42: Diagrama secuencia plotDay

PlotDay muestra en una gráfica la variación de iluminación durante el transcurso de un día. El código utilizado es el siguiente:

```
#include <iostream>
#include <vector>
#include <boost/progress.hpp>

#include "../../base/include/sgps.h"
#include "../../base/include/plotter.h"

using namespace std;

int main (int argc, char* argv[]) {
»
»
»       SGPS::parseArguments(argc, argv);
»
»       DirAnalyzer dir;
»
»       Day d = dir.extractDay(argv[1]);
»
»
»       d.plotDay();
»
»
»
»       return 0;
}
```

Figura 43: Plot Day

La función de este ejemplo es la de saber si al indicar el tipo de archivo a leer realiza la gráfica correcta. Para ello se probará un ejemplo con cada uno de ellos.

Lo primero que se debe hacer al ejecutar el programa es indicar la ruta del archivo seguido de *-f* y el tipo. Para el primer ejemplo utilizaremos un archivo de tipo genérico como se ve en la gráfica 44

```
isaac@isaac-K55VD:~/Dropbox/Proyecto/sgpsproject-trunk/SGPS_CPP/build/bin$
./sgps_plotday ../../../data/genericmeasures.txt -f generic
```

Figura 44: Ejecución de plotDay con archivo genérico

Una vez se le indica la ruta y el tipo, se creará una objeto del tipo deseado y se extraerá toda la información de los datos de iluminación y horas, mandándolos después en un vector para que sean representados. La salida del programa es la indicada en la figura 45

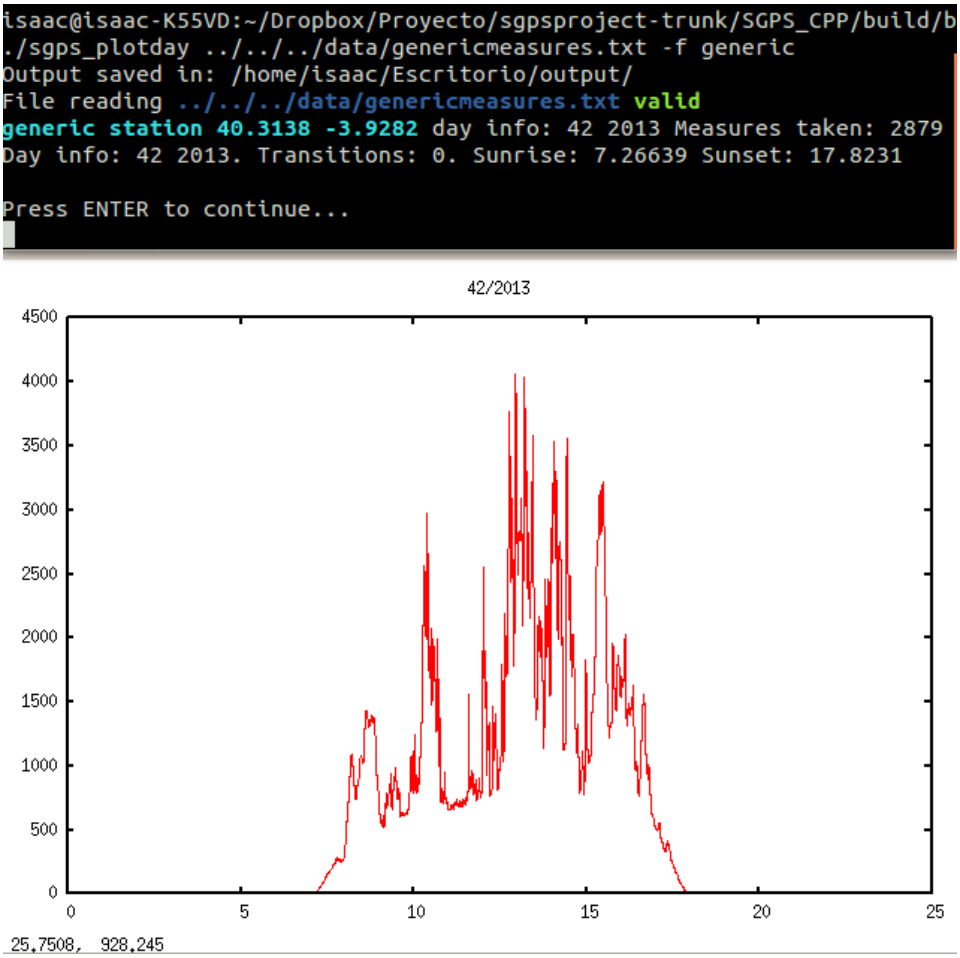

Figura 45: Salida de plotDay con archivo genérico

Los ejemplos para un archivo NOAA o HUEFile son iguales a estas y se pueden ver en las figuras 46 y 47.

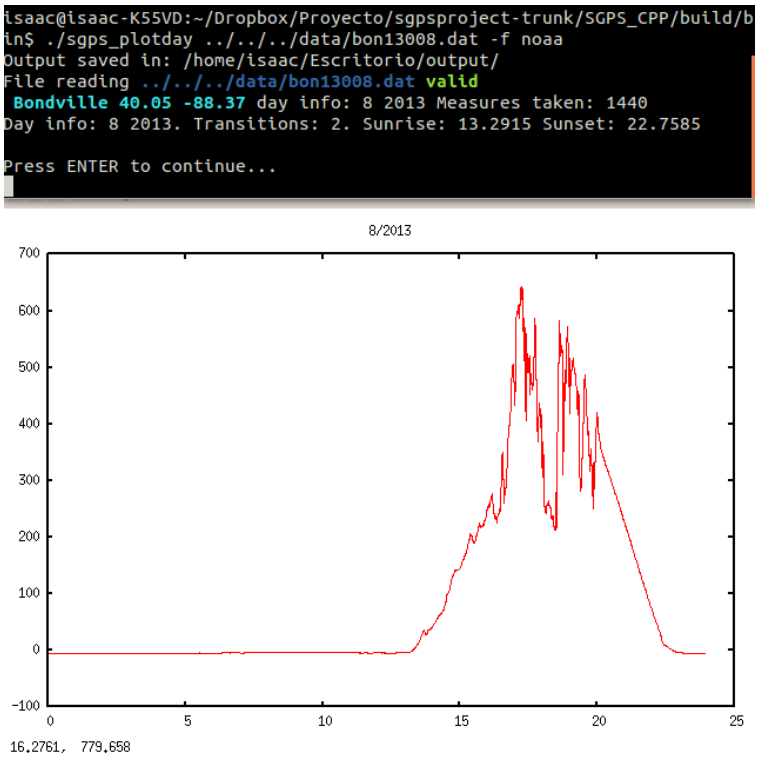

Figura 46: Salida de plotDay con archivo NOAA

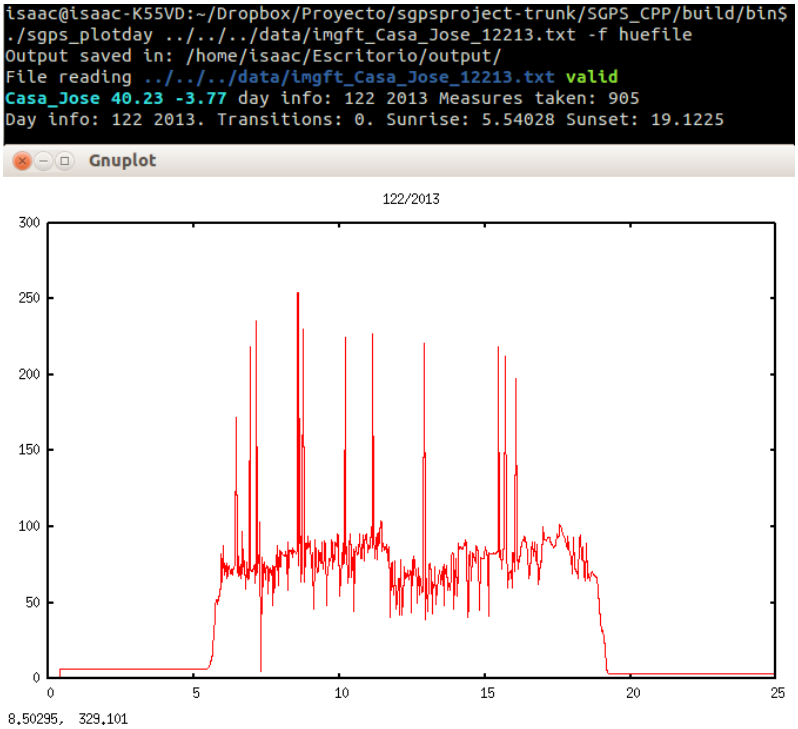

Figura 47: Salida de plotDay con archivo HUE

5.3. Plot Coordinates

El diagrama de este ejemplo y su código son los mostrados en las siguientes figuras.

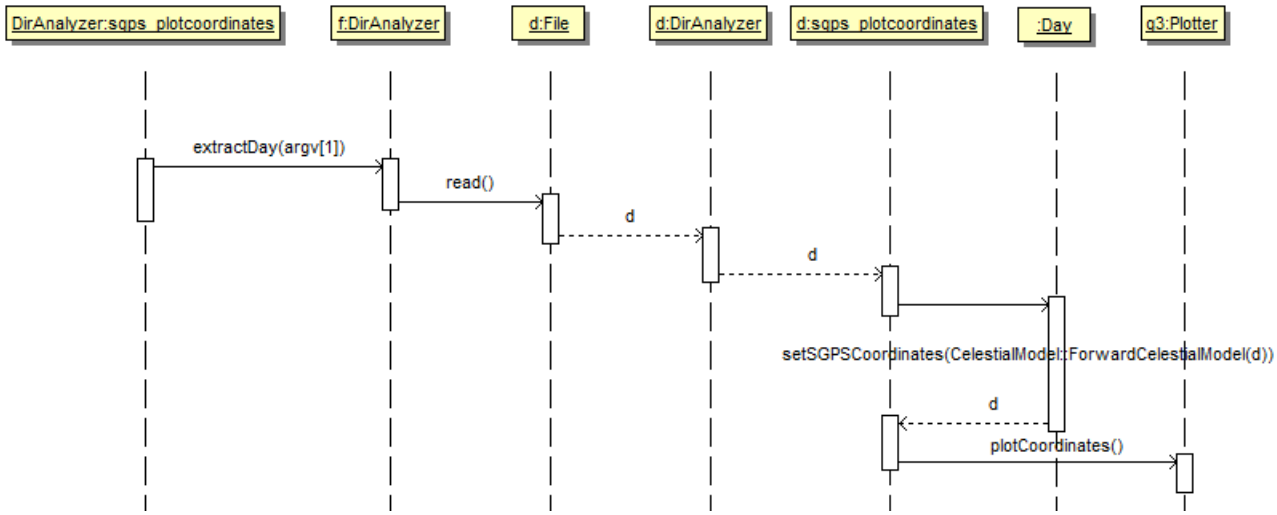

Figura 48: Diagrama secuencia plotCoordinates

```
#include <iostream>
#include <vector>
#include <boost/progress.hpp>


#include <sgps/sgps.h>



using namespace std;

int main (int argc, char* argv[]) {
»
»       SGPS::parseArguments(argc, argv);
»
»       DirAnalyzer dir;»
»
»       Day d = dir.extractDay(argv[1]);
»
»       d.setSGPSCoordinates(CelestialModel::ForwardCelestialModel(d));
»
»       Plotter::plotCoordinatesDay(d, argv[2]);
»
»       return 0;
}
```

Figura 49: Plot Coordinates

Como puede observarse la biblioteca diseñada viene incluida entre '<>', lo que indica que la librería esta instalada en el equipo; esto se tratará detenidamente más adelante. Este ejemplo fue diseñado para ver si la librería estaba bien instalada, y su única función es la de representar en una gráfica la comparación de las coordenadas

reales con las calculadas. Al igual que con plot day se hará un ejemplo con cada tipo de archivo. El resultado es el siguiente:

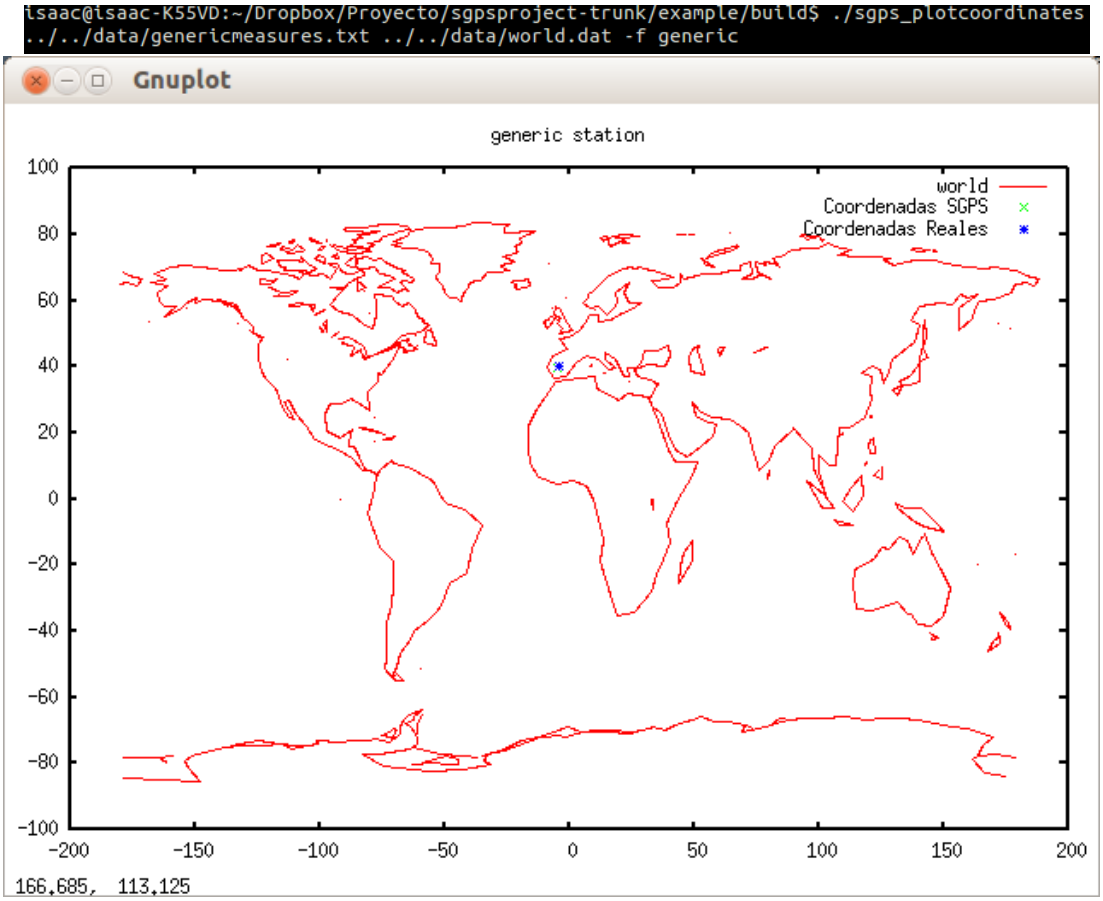

Figura 50: Ejemplo de Plot Coordinates con archivo genérico

Como se ve en la figura la librería funciona como debe, y se observa como el modelo usado por SGPS para el cálculo de las coordenadas es bastante fiable.

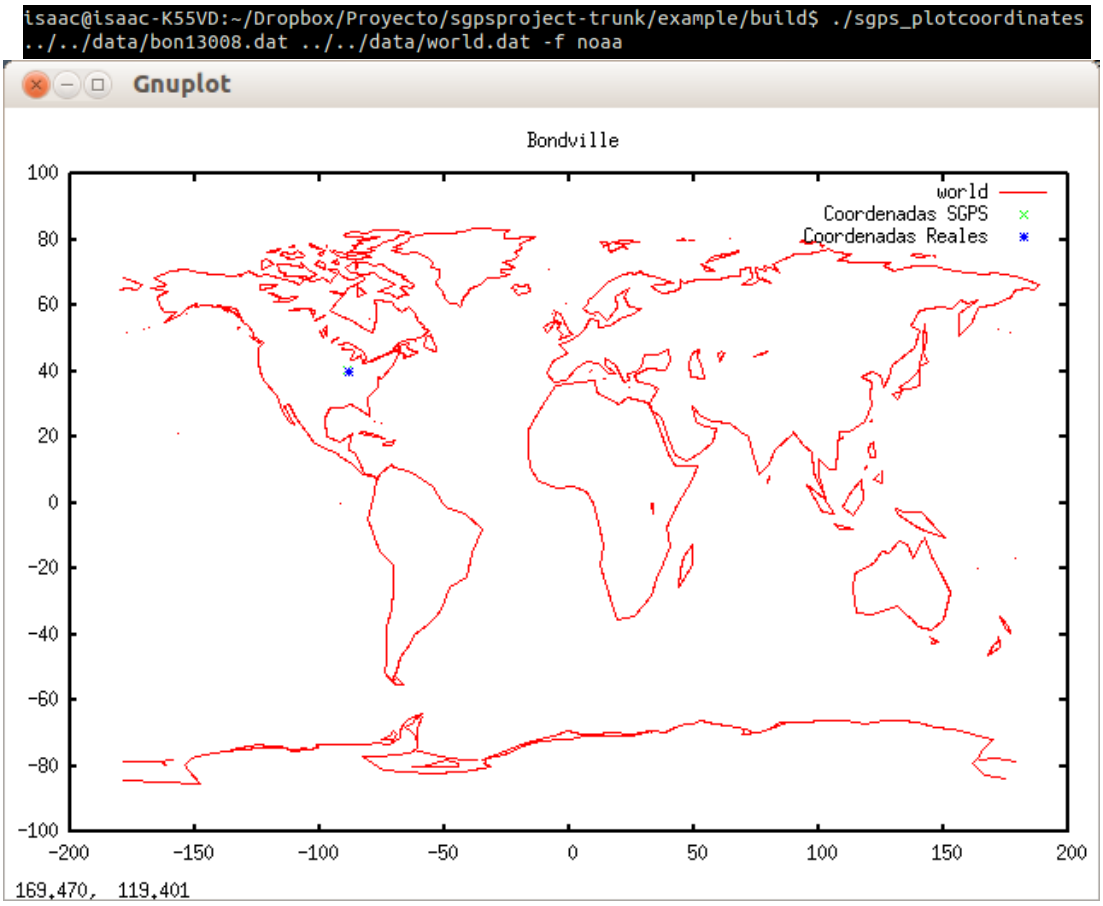

Figura 51: Ejemplo de Plot Coordinates con archivo NOAA

Al igual que con un archivo genérico, la librería muestra ambas coordenadas correctamente.

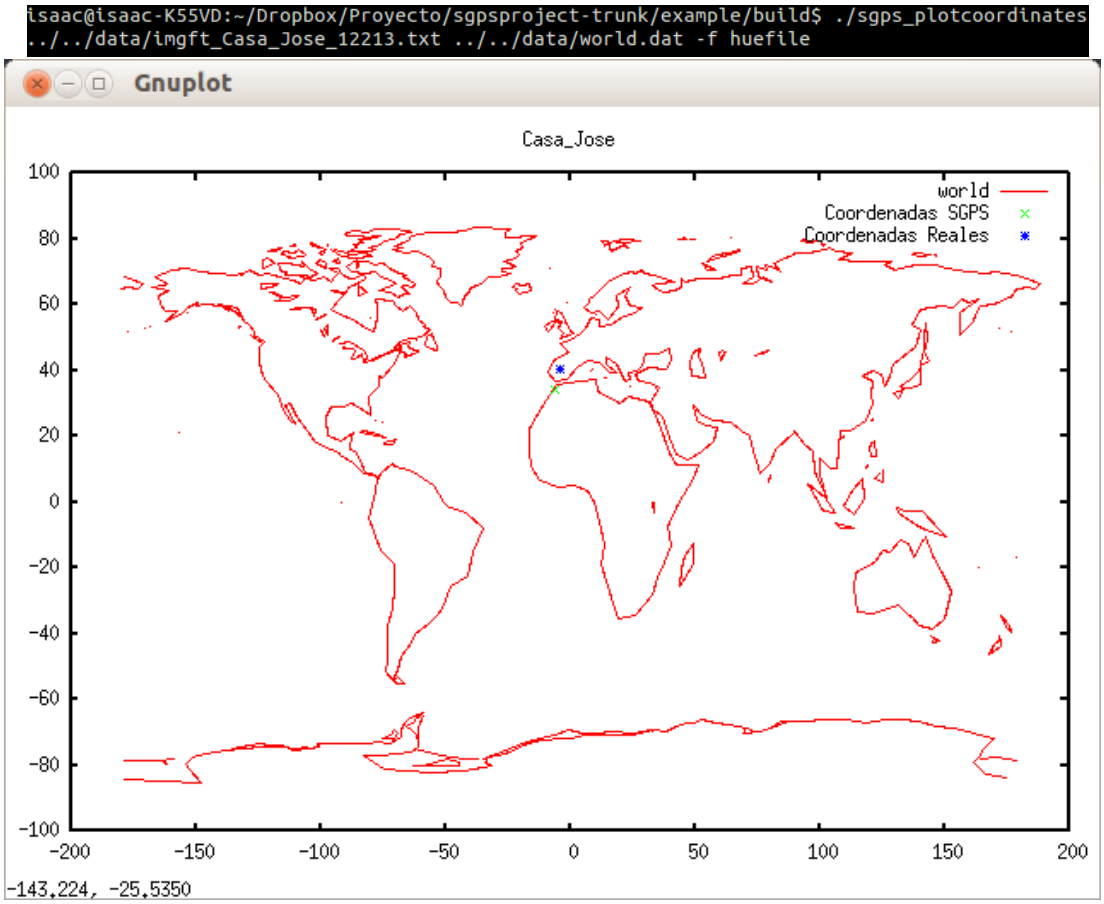

Figura 52: Ejemplo de Plot Coordinates con archivo HUE

En este ejemplo también se ve como la biblioteca funciona a la perfección, salvo que el cálculo de las coordenadas no es óptimo. Como en los anteriores ejemplos hallaba perfectamente las coordenadas, el problema de este tipo de archivos es a la hora de tomar las medidas.

6. Información avanzada para desarrolladores

6.1. Añadir nuevos tipos de archivos

Esta subsección explica como añadir un nuevo tipo de extracción de datos para permitir la utilización de archivos propios, que no se ajusten a los que se han explicado. Para ello se utilizará a modo de ejemplo la clase HUEFile.

Los archivos HUEFile salen de un script de la herramienta de desarrollo MATLAB. Este script, a través de una webcam orientada a la calle, hace fotos durante un día entero. A partir de esas fotos obtiene los parámetros de iluminación, contraste y modelos de color RGB, HSV y LAB, almacenándolos luego en un archivo .txt. Para el modo de lectura de esta clase hay que tener en cuenta que las coordenadas no sólo pueden obtenerse con la iluminación, sino también con ciertos parámetros de los modelos de color. Por ello en este modo de extracción de datos se deberá tener en cuenta la columna que se va a utilizar.

Lo primero que hay que tener en cuenta es la forma en la que se van a distinguir los archivos de este tipo del resto. Un HUEFile llevará siempre al comienzo del nombre *imgft*, por lo que primero se comprobará esto. Esta comparación debe colocarse en la función *getValidFiles()* de DirAnalyzer. El código utilizado es el siguiente:

```
if (strcmp (s, "genericfile") == 0){

    File * filePtr = new File(it->getPath());
    validFiles.push_back(filePtr);
}

else if (strncmp (name.c_str(), "imgftxx", 5) == 0){

    File * filePtr = new HUEFile(it->getPath());
    validFiles.push_back(filePtr);
}
```

Figura 53: Código para distinguir HUEFile

Mediante la función *strncmp()* se compara el nombre del archivo con *imgft*. En caso de que coincidan se creará una instancia a un tipo de archivo HUEFile, añadiéndolo al vector de ficheros válidos.

También se deberá tener en cuenta que se podía indicar el tipo de archivo que era al comienzo del programa con la opción *-f*. Para ello habrá que colocar el código descrito en 54 en la función *extractDay* también de DirAnalyzer.

```
else if (opt->isFileType("huefile"))
    f = new HUEFile(path);
```

Figura 54: Código para distinguir HUEFile

En esta parte se comprueba si después de la opción *-f* aparece *huefile*. Si lo hiciera se crearía un objeto del tipo HUE que más adelante se mandaría a la clase HUEFile para extraer sus datos. Una vez hecho esto se pasa a la creación de la clase. Se deberá crear un constructor específico que contenga lo siguiente.

```
#ifndef HUEFILE_H_
#define HUEFILE_H_


#include <vector>
#include <boost/algorithm/string.hpp>
#include <boost/filesystem.hpp>
#include "day.h"
#include "file.h"



class HUEFile: public File {
public:
	HUEFile();
	HUEFile(const boost::filesystem::path);
	virtual ~HUEFile();
	
	
	
	
	Day read (bool saveData = false);
	
	
	
private:
	
	int elevation;
	int column;
	
};

#endif /* HUEFILE_H_ */
```

Figura 55: huefile.h

```
HUEFile::HUEFile() {

}

HUEFile::HUEFile(const path p)
{

    name  = p.filename().string();
    ppath = p.string();
    size  = file_size(p);
    column = 4;
    sslimit = 10;
}


HUEFile::~HUEFile() {

}
```

Figura 56: Constructor de HUEFile

Como se ve en la foto al crear el objeto HUEFile se guardarán el nombre, la ruta, el tamaño y las variable *column* y *sslimit*. La columna se colocará la número 4 por defecto, pero se le puede indicar otra. *Sslimit* indica el límite que se toma para el amanecer y el anochecer cuando se busque la hora de salida o puesta del sol. Con el constructor definido se deben crear las funciones que tendrá esta clase. En este caso tan solo se modificará la forma de leer archivos, es decir la función *read()*. Estos archivos son del tipo que aparece en la figura 57

```
3.77    40.23   Casa_Jose       2       4       2013    648
 0       0       24      1.01    0.86    0.01    0.01    0.00    0.99    1.02    0.98    3.59    -0.19   0.18
 0       1       28      1.01    0.79    0.01    0.02    0.00    0.99    1.02    0.98    3.59    -0.18   0.17
 0       2       31      1.01    0.79    0.01    0.01    0.00    0.99    1.03    0.98    3.59    -0.19   0.17
 0       3       34      1.01    0.79    0.01    0.01    0.00    0.99    1.02    0.98    3.59    -0.18   0.18
 0       4       38      1.01    0.79    0.01    0.01    0.00    0.99    1.02    0.98    3.59    -0.19   0.18
 0       5       41      1.01    0.83    0.01    0.02    0.00    0.99    1.03    0.98    3.59    -0.18   0.17
 0       6       45      1.01    0.86    0.01    0.02    0.00    0.99    1.02    0.98    3.59    -0.19   0.17
 0       7       48      1.01    0.90    0.01    0.01    0.00    0.99    1.03    0.98    3.59    -0.18   0.17
 0       8       52      1.01    0.86    0.01    0.01    0.00    0.99    1.03    0.98    3.59    -0.19   0.17
 0       9       55      1.01    0.75    0.01    0.02    0.00    0.99    1.03    0.98    3.59    -0.18   0.17
 0       10      59      1.01    0.86    0.01    0.02    0.00    0.99    1.02    0.98    3.59    -0.18   0.18
 0       12       2      1.01    0.83    0.01    0.02    0.00    0.99    1.03    0.98    3.59    -0.18   0.17
 0       13       6      1.01    0.75    0.01    0.01    0.00    0.99    1.02    0.98    3.59    -0.19   0.18
 0       14       9      1.01    0.86    0.01    0.02    0.00    0.99    1.02    0.98    3.59    -0.18   0.17
 0       15      13      1.01    0.83    0.01    0.01    0.00    0.99    1.03    0.98    3.59    -0.19   0.17
 0       16      17      1.01    0.90    0.01    0.01    0.00    0.99    1.03    0.98    3.59    -0.19   0.18
 0       17      20      1.01    0.75    0.01    0.02    0.00    0.99    1.02    0.98    3.59    -0.18   0.17
 0       18      24      1.01    0.75    0.01    0.01    0.00    0.99    1.02    0.98    3.59    -0.19   0.17
 0       19      28      1.01    0.83    0.01    0.01    0.00    0.99    1.02    0.98    3.59    -0.19   0.18
 0       20      31      1.01    0.75    0.01    0.02    0.00    0.99    1.02    0.98    3.59    -0.18   0.17
 0       21      35      1.01    0.90    0.01    0.02    0.00    0.99    1.03    0.98    3.59    -0.18   0.17
```

Figura 57: Archivo HUE

Los parámetros que aparecen en la primera línea son longitud, latitud, lugar, día, mes, año y elevación. A partir de la segunda línea las columnas son hora, minuto, segundo, iluminación, contraste y valores de los modelos RGB, HSV y LAB. La función *read()* para este caso deberá ser de la siguiente manera:

```
Day HUEFile::read (bool saveData) {

	Options * opt =  Options::Instance();

	Day d;
	fstream stored;

	int col = opt->getColumnNumber();
	if (col > 4)
		column = col;

	switch (column){
		case 4: sslimit = 9;
			break;
		case 5: sslimit = 0.9;
			break;
		case 6: sslimit = 0.3;
			break;
		case 7: sslimit = 0.1;
			break;
		case 8: sslimit = 0.1;
			break;
		case 9: sslimit = 10;
			break;
		case 10: sslimit = 20;
			break;
		case 11: sslimit = 30;
			break;
		case 12: sslimit = 15;
			break;
		case 13: sslimit = 0.1;
			break;
		case 14: sslimit = 0.1;
			break;
	}

	stored.open(ppath.c_str(), fstream::in);

```

Figura 58: Función *read()* HUEFile

Lo primero es crear el objeto de la clase Day que se debe devolver al final de la función. La parte siguiente de la función solo es para colocar el límite de del amanecer y anochecer según la columna escogida, y no debe implementarse en caso de que no se vayan a elegir columnas. Terminado este proceso se abre el archivo.

```
if (stored.good()) {

        float latitude, longitude;
        stored >> longitude >> latitude;
        Coordinates c;
        c.latitude = latitude;
        c.longitude = longitude;
        d.setRealCoordinates(c);


        string s;
        stored >> s;
        d.setStation (s);

        //The file pointer is moved to the day position.
        int day, moth;
        stored>>day>>moth;
        if (moth==3){
                day = day +28+31;
        }
        else if (moth<=7 && moth%2==0 && moth !=2){
                day = day + (moth/2)*31 + (moth/2-1)*30 + 28;
        }
        else if (moth>7 && moth%2==0){
                day = day + (moth/2+1)*31 + (moth/2-2)*30 + 28;
        }
        else if (moth<=7 && moth%2!=0 && moth !=1 && moth !=3){
                day = day + (moth/2-1)*30 + (moth/2+1)*31 + 28;
        }
        else if (moth>7 && moth%2!=0){
                day = day + (moth/2-1)*30 + (moth/2+1)*31 + 28;
        }
        else if (moth == 1){
                day = day;
        }
        else if (moth == 2){
                day = day + 31;
        }
        d.setDay(day);
```

Figura 59: Función *read()* HUEFile

Con el archivo abierto se pasa a la extracción de datos, para lo cual utilizamos la función *stored*. Se extraerán los parámetros y se guardaran por orden en la variables: longitud, latitud, lugar, día, mes (que deberá convertirse en un día de 1 a 365) y año. Una vez guardados se enviarán al objeto 'd' creado mediante funciones *set()*.

```
//The file pointer is moved to the year position.
int year;
stored >> year;
d.setYear(year);


stored >> elevation;

float hour, min, seconds, auxvalues, transitions=0;
float a;

while(!stored.eof())
{

	stored >> hour >> min >> seconds;

	hour = hour + min/60;
	hour = hour + seconds/3600;//pass seconds to hours

	times.push_back(hour);


	for (int i=4; i<column; i++){
		stored >> a;
	}

	stored >> auxvalues;
	values.push_back(auxvalues);


	for(int z=column; z<14; z++){
		stored >> a;
	}

}
```

Figura 60: Función *read()* HUEFile

Una vez extraídos todos los parámetros de información deberán guardarse los parámetros de iluminación junto con sus horas. Para ello se especificará que mientras no se llegue al final del archivo se deben guardar en un vector las horas (teniendo en cuenta que debe ser de 0 a 24 h) y la columna deseada. Para esto último, se hacen dos bucles *for* desechando las columnas que no se necesiten.

```
File::setSkipFile(0);
d.setTransitions(0);
float sunrise = File::findSunrise(times, values);
float sunset = File::findSunset(times, values);
d.setSunrise(sunrise);
d.setSunset(sunset);

// Not saved by default (consumes time and a lot of memory).
if (saveData) {
	d.setTimes (times);
	d.setValues (values);
}


if (opt->getConsoleOp() == true || opt->getFileReadedOp() == true){
	Console::validFiles (name, ppath, File::getSkipFile());
}
if (opt->getConsoleOp() == true){
	Console::placeInfo(d, values.size());
}
if (opt->getLoggerOp() == true){
	Logger::placeInfo(d, values.size());
	Logger::validFiles(name, ppath, File::getSkipFile(), d);//save
}

return d;
```

Figura 61: Función *read()* HUEFile

Una vez termina el fichero, se calcula el *sunrise* y *sunset*, y en el caso de que se vayan a calcular más adelante las coordenadas se mandan los vectores a la clase Day. Ese envío no es directo para no utilizar memoria innecesaria, ya que en la librería no es necesario siempre el cálculo de las coordenadas; por ejemplo en Plot Day 5.2 no hace falta guardar estos vectores, porque tan solo se representa la variación de iluminación en un día. Marcando las respectivas opciones es posible llamar a las funciones de Logger, Console y Plotter.

6.2. Añadir nuevas opciones

Esta sección mostrará como añadir nuevas opciones a las ya creadas (-l, -c, -p, etc.). Para ello se utilizará de ejemplo la opción para elegir columna de la clase HUEFile, que se activará si se pasa un *-h* seguido del número de columna; en caso de no proporcionárselo, saldrá un error por pantalla. Para crear esta opción se deberá colocar en la función *parseArguments()* de SGPS lo siguiente:

```
vector<int> vals;
index = Console::parseArguments(argc, argv, "-h", vals);
if (index > 1)
	if (vals.size() == 0){// Option but no values.
		Console::error("Parameters for option -h (column number) not introduced.");

	}
	else
		for (vector<int>::iterator it (vals.begin()); it != vals.end(); ++it)
			opt->setColumnNumber(*it);
```

Figura 62: Nueva opción en *parseArguments()*

Esta función manda todos los argumentos junto con un vector vacío, para guardar el número de la columna, y la opción requerida, para compararla, a la función *parseArguments()* de Console. En ella se comprueba que se ha pasado la opción con la función *findArguments()* y se guardan en un vector todos los números que aparezcan detrás de *-h*, hasta que vuelva a aparecer un '-', que indicaría que viene otra opción.

```
int Console::parseArguments (int argc, char** argv, const char* str, vector<int> & vals)
{
  int index = findArguments (argc, argv, str);
  int i = index;

	string s;
	int val;
	do {
		i++;
		if (i > argc - 1) // Checking the limits
			s = '-';
		else {
			val =atoi(argv[i]);
			vals.push_back(val);
		}
	} while (s[0] != '-');

  return index;
}
```

Figura 63: Nueva opción en *parseArguments()*

Además, se ha tenido en cuenta que si solo se le pasara esta opción por pantalla deberían activarse las demás funciones por defecto (Logger, Console y Plotter). Esto se puede hacer, ya que se sabe que habrá cuatro argumentos si se activara solo la opción *-h* con un valor de columna, y tres con la opción *-h* sin valor de columna.

Para comprobar el funcionamiento se realizará un ejemplo con Plot Day. En el primero se pasará la opción con la columna 8. En el segundo se pondrá la opción sin columna, por lo que deberá aparecer un mensaje de error y tomará la opción por defecto, 4.

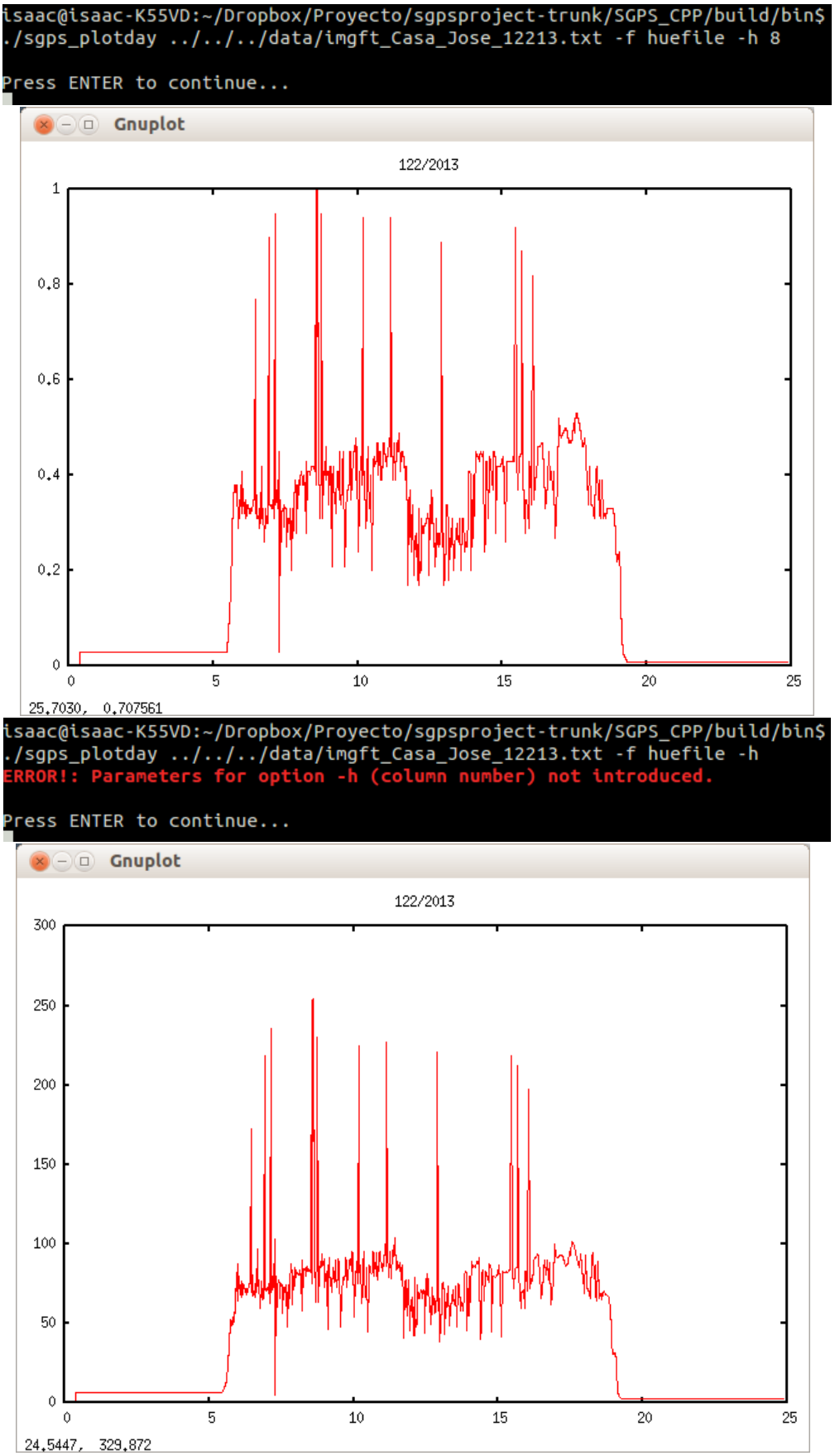

Figura 64: Ejemplo de nueva opción en *parseArcguments()*

7. Conclusión y trabajos futuros

En esta sección se van a analizar las conclusiones del proyecto, las limitaciones que puede tener y los trabajos futuros que pueden crearse a partir de esta librería.

Los objetivos del trabajo se han cumplido con éxito: se ha creado una librería que utiliza el modelo del SGPS y que, al estar ordenada por clases bien diferenciadas donde cada una tiene su función definida, es fácil de manejar. Además, gracias a este proyecto, se han adquirido muchos conocimientos de programación y diseño, que ayudan a tener una idea de como estructurar un proyecto de este estilo para el futuro.

Con respecto a las líneas futuras de investigación cabe decir que son muy amplias, ya que al ser una librería se le pueden añadir nuevas funciones actualmente no existentes, como por ejemplo desarrollar un nuevo algoritmo para el cálculo de coordenadas. También se pueden crear otras clases para nuevos tipos de archivos o mejorar las ya utilizadas, como es el caso de la clase gnuplot, ya que una librería nueva podría solucionar algunos de los problemas que se han tenido en el diseño. También se puede intentar corregir el problema que se tiene en el cálculo de la latitud o mejorar la forma de encontrar la salida y la puesta del sol.

Esta librería servirá de base para futuros proyectos, algunos de los cuales ya han comenzado su desarrollo: la inclusión en el sistema de diferentes magnitudes como la la presión atmosférica o la luz va camino de convertirse en realidad, y la clase HUEFile es la mejor prueba de ello.

A. Utilización de la librería

En esta sección se explica como usar la librería.

Los requisitos básicos son tener el sistema operativo Linux, ya que por el momento las rutas para Windows y Mac OS X no están diseñadas, y tener instalado el programa gnuplot. Si esto se cumple, desde la carpeta *build* se abre el terminal y se escribe el comando "*cmake ..*". Este comando leerá el archivo CMakeList.txt y compilará toda la librería.

```
isaac@isaac-K55VD:~/Dropbox/Proyecto/sgpsproject-trunk/SGPS_CPP/build$ cmake..
```

Figura 65: Compilar librería

Una vez compilado hay dos opciones: instalar la librería y los ejemplos en el equipo o sólo construirlos.

A.1. Construcción con instalación

Si se desea instalar la librería en el equipo el Cmake debe llevar los siguientes comandos:

```
INSTALL(PROGRAMS ${PROJECT_BINARY_DIR}/bin/sgps_sgpstest ${PROJECT_BINARY_DIR}/bin/sgps_plotday
  DESTINATION bin)


INSTALL(TARGETS sgps
  RUNTIME DESTINATION bin
  LIBRARY DESTINATION lib
  ARCHIVE DESTINATION lib
)


INSTALL(FILES src/base/include/sgps.h
              src/base/include/astroalg.h
              src/base/include/coordinates.h
              src/base/include/celestialmodel.h
              src/base/include/console.h
              src/base/include/day.h
              src/base/include/diranalyzer.h
              src/base/include/file.h
              src/base/include/noaafile.h
              src/base/include/gnuplot_i.h
              src/base/include/logger.h
              src/base/include/options.h
              src/base/include/plotter.h
              src/base/include/huefile.h
  DESTINATION include/sgps)
```

Figura 66: CmakeList para instalar librerías

Para instalarla, teniendo el terminal abierto desde *build* se escribirá *sudo make install* seguido de la contraseña de superusuario. Este comando proporciona derechos de superusuario (*sudo*), y construye e instala todo lo necesario.

```
isaac@isaac-K55VD:~/Dropbox/Proyecto/sgpsproject-trunk/SGPS_CPP/build$
sudo make install
[sudo] password for isaac:
```

Figura 67: Instalación librería

Una vez instalada, para utilizar los ejemplos descritos tan sólo hará falta poner en nombre y la ruta en el terminal de la siguiente manera:

```
isaac@isaac-K55VD:~$ sgps_sgpstest Escritorio/Measures
```

Figura 68: Ejecución de SGPS Test

Si se desea crear un nuevo ejemplo teniendo la biblioteca instalada se debe crear un CMakeList.txt que contenga lo siguiente:

```
PROJECT(SGPSProject)

SET(CMAKE_BUILD_TYPE Debug)


INCLUDE_DIRECTORIES(/usr/local/include/)

#It could be necessary to change this path
#LINK_DIRECTORIES(/usr/lib/ )

SET(SGPSProject_SRCS
»       »       »       Nombre_archivo.cpp
                )

ADD_EXECUTABLE(Nombre_ejecutable ${SGPSProject_SRCS})

TARGET_LINK_LIBRARIES(Nombre_ejecutable sgps boost_filesystem boost_system m)
```

Figura 69: CMakeList para ejemplos con instalación

Una vez compilado y construido, si se desea ejecutar, debe escribirse *./Nombre_ejecutable* y los argumentos necesarios desde el terminal en la carpeta del ejecutable.

A.2. Construcción sin instalación

Si no se quiere instalar la librería y sólo se va a utilizar, en vez del comando anterior debe escribirse *make*.

```
isaac@isaac-K55VD:~/Dropbox/Proyecto/sgpsproject-trunk/SGPS_CPP/build$ make
```

Figura 70: Construcción sin instalación

Para ejecutar se debe abrir un terminal en la carpeta *bin* que se ha creado en *build* y escribir en él *./sgps_sgpstest* y los argumentos necesarios.

```
isaac@isaac-K55VD:~/Dropbox/Proyecto/sgpsproject-trunk/SGPS_CPP/build/bin$ ./sgps_sgpstest
../../../data
```

Figura 71: Ejecución SGPS Test

En el caso que se quiera crear un ejemplo, se debe poner en la carpeta *apps*, dentro de *src*, con la misma distribución que los ejemplos un CMakeList.txt que contenga únicamente lo descrito en la figura 72.

```
SGPS_ADD_EXECUTABLE("AUTO" "ALL_FILES_RECURSE")

# If you need to link some other external library:
TARGET_LINK_LIBRARIES(${EXE_NAME} boost_filesystem boost_system m)
```

Figura 72: CMakeList para ejemplos sin instalación

Además se debe añadir el subdirectorio en el CMakeList de la carpeta *apps*. Para la ejecución se procederá igual que en los ejemplos descritos.

B. Presupuesto

En el presente anexo se calculará el coste del valor del proyecto presentado a lo largo del documento, detallando cada coste por separado.

B.1. Costes de personal

Los costes de personal se han separado en dos tipos diferentes: tipo de personal y apartado del proyecto. La primera división obedece a la diferencia salarial entre el ingeniero y el director del proyecto, mientras que la segunda división detalla las horas empleadas en cada sección del proyecto, dada la magnitud de éste.

Para la división del proyecto en apartados se ha seguido la siguiente estructura:

- Planteamiento. Incluye la familiarización con las librerías necesarias, el estudio del algoritmo inicial y el diseño general del proyecto.
- Desarrollo. Incluye varios apartados, entre los que se encuentran el desarrollo del algoritmo y la librería y la investigación requerida para las distintas partes que lo componen.
- Redacción de la memoria. Incluye tanto la redacción como la corrección de la misma.

		Director de proyecto	Ingeniero
Planteamiento	Familiarización (h)	0	20
	Estudio del algoritmo (h)	10	20
	Diseño (h)	5	15
Desarrollo	Desarrollo del algoritmo y librería (h)	5	300
	Investigación (h)	5	150
Memoria	Redacción (h)	0	60
	Corrección (h)	20	20
Total	Total (h)	45	585
	Salario/hora	50	30
	Salario total (€)	2250	17550
	Total (€)		19800

Tabla 1: Costes desglosados de personal

B.2. Costes materiales

Los costes materiales incluye todo lo relacionado con los costes de hardware y software y recursos de oficina. Dentro del hardware se incluye un ordenador portátil de gama alta en el que poder realizar la búsqueda de información y todas

las tareas relacionadas con el desarrollo de la librería. Este dispositivo tiene un plazo de amortización de tres años, y se han tenido en cuenta los 4 meses de duración del proyecto. En cuanto al software, se han utilizado alternativas libres y gratuitas tanto en el sistema operativo como en los programas y librerías, por lo que el coste es nulo. Finalmente, respecto a los recursos de oficina únicamente se tiene en cuenta el coste de acceso de Internet.

			Coste de proyecto
Hardware	Ordenador	800 (€total)	88.89
Software	Software	0	0
Oficina	Internet	30 (€/mes)	120
Total (€)			208.89

Tabla 2: Costes desglosados de material

B.3. Total

Sumando los apartados anteriores y añadiendo tanto costes indirectos (veinte por ciento) como IVA (veintiuno por ciento), el precio total del proyecto asciende a *veintinueve mil cincuenta y dos euros con noventa y un céntimos*

Se he tenido en cuenta que todas las librerías de apoyo usadas son de libre distribución y código abierto, por lo que éstas no se han tenido en cuenta en la realización del presupuesto. De ser así, ya que son horas de trabajo por programadores, diseñadores y testers, el presupuesto del proyecto ascendería a varios cientos de miles de euros.

	Coste(€)
Personal	19800
Materiales	208.89
Costes indirectos (20 %)	4001.78
Subtotal	24010.67
IVA (21 %)	5042.24
Total (€)	29052.91

Tabla 3: Costes desglosados totales

Leganés, 24 de Junio de 2013

El ingeniero

Referencias

[1] SGPS project. Algoritmo para localizar un lugar mediante una foto. *SGPS Project Server*. Online: `www.sgpsproject.org/index.php/History`. Última visita: 8 de abril 2013.

[2] Gomez, Javier V.; Sandnes, Frode Eika; Fernández, Borja, Sunlight Intensity Based Global Positioning System for Near-Surface Underwater Sensors, *MDPI Journal*, Febrero 2012, Volumen 12(2), páginas 1930-1949.

[3] CIESE, Definición de paralelos y meridianos, *CIESE, The Center for Innovation in Engineering and Science Education*. Online: `http://ciese.org/ciberaprendiz/latylong/merypar.htm`. Última visita: 5 de mayo de 2013

[4] Wikipedia, Coordenadas geográficas, *Wikipedia, la enciclopedia libre*. Online: `http://es.wikipedia.org/wiki/Coordenadas_geograficas`. Última visita: 8 de abril 2013.

[5] Wikipedia, Coordenadas esféricas, *Wikipedia, la enciclopedia libre*. Online: `http://es.wikipedia.org/wiki/Coordenadas_esfericas`. Última visita: 8 de abril 2013.

[6] Definición de latitud, *Mathematicts Dictionary*. Online: `http://www.mathematicsdictionary.com/spanish/vmd/full/l/latitude.htm`. Última visita: 9 de abril 2013.

[7] Definición de longitud. *Mathematicts Dictionary*. Online: `http://www.mathematicsdictionary.com/spanish/vmd/full/l/longitude.htm`. Última visita: 9 de abril 2013.

[8] Stroustrup, Bjarne, *Bjarne Stroustrup Home page*. Online: `http://www.stroustrup.com/`. Última visita: 7 de mayo de 2013.

[9] Wikipedia, gnuplot, *Wikipedia, la enciclopedia libre*. Online:`http://es.wikipedia.org/wiki/Gnuplot`. Última visita: 7 de mayo de 2013.

[10] Stallman, Richard, *Richard Stallman Personal Site*. Online: `http://stallman.org/`. Última visita: 7 de mayo de 2013.

[11] Documetación de doxygen, *Doxygen Server*. Online:`http://www.stack.nl/~dimitri/doxygen/`. Última visita: 7 de mayo de 2013.

[12] Documentación CMake, *CMake Server*. Online: `http://www.cmake.org`. Última visita: 7 de mayo de 2013.

[13] Wikipedia, Librería, *Wikipedia, la enciclopedia libre*. Online: `http://en.wikipedia.org/wiki/Library_(computing)`. Última visita: 8 de mayo de 2013.

[14] Conlin, Jeremy. En *Google Code*. Online: `https://code.google.com/p/gnuplot-cpp/`. Última visita: 8 de mayo de 2013.

[15] Alexander, Christopher, *A Pattern Languages (El lenguaje de los patrones)*, 1977.

[16] SGPS Library, Rivero, Isaac; Gomez, Javier V. *SourceForge*. Online: `http://sourceforge.net/p/sgpsproject/trunk/ci/master/tree/`. Última visita: 19 de mayo de 2013.